09/10/2021

Data Structure and Algorithms

Applied Prep Course

SATYAM SETH

PART-1

Sorting

Page No.: 1
Date: 18/10/20

- **Satellite data and Key.-**

Eg- Sort record according to their price.

id	name	price	desc.
3	n_3	P_3	d_3
1	n_1	P_1	d_1
5	n_5	P_5	d_5

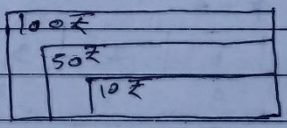
Where $P_3 < P_1 < P_5$

Here P_3, P_1 P_5 are the key and id, name, desc are satellite data which are move according to the price.

Note- ascending order is default order of sorting.

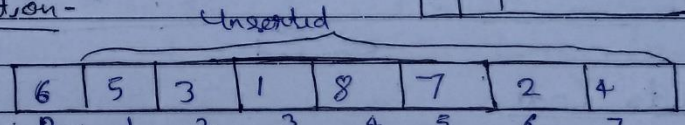
- **Insertion Sort -**

→ **Intution-**



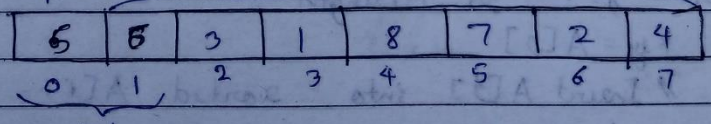
Sorting notes is an example of insertion sort

Eg-



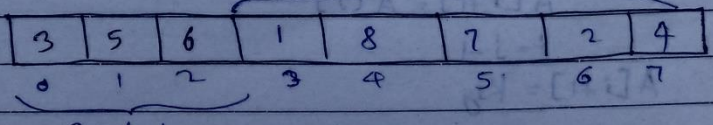
Sorted

K = 5

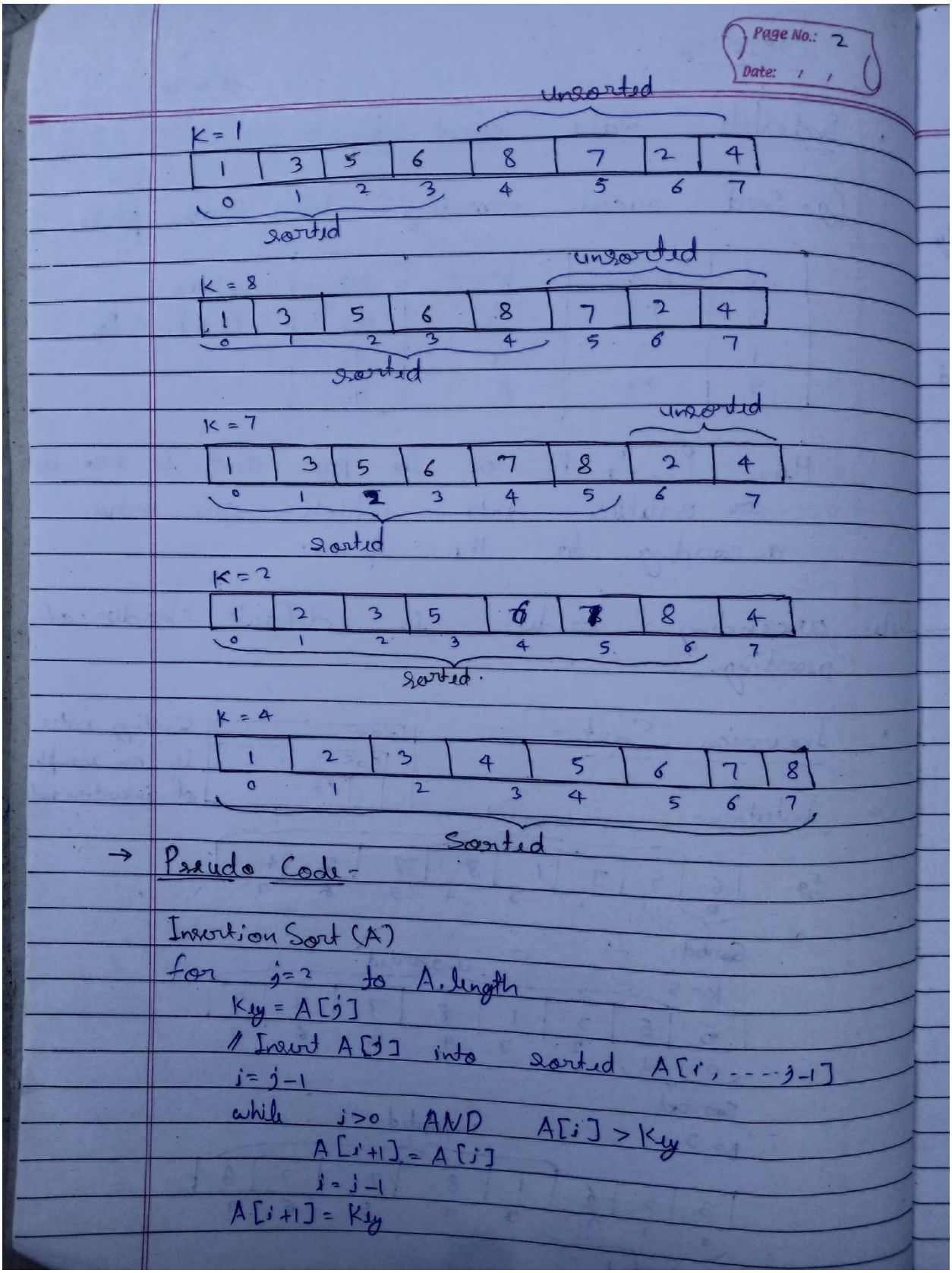


Sorted

K = 3



Sorted



→ Pseudo Code -

```
Insertion Sort (A)
for j=2 to A.length
    Key = A[j]
    // Insert A[j] into sorted A[i, ---- j-1]
    i = j-1
    while i>0 AND A[i] > Key
        A[i+1] = A[i]
        i = i-1
    A[i+1] = Key
```

Page No.: 3
Date: / /

→ Code -

```
void InsertionSort(int arr[], int i)
{
    for(int j=1; j<n; j++)
    {
        int key = arr[j];
        int i = j-1;
        while(i>=0 && arr[i] > key)
        {
            arr[i+1] = arr[i];
            i--;
        }
        arr[i+1] = key;
    }
}
```

Note - Insertion Sort is "In-place Sorting" algorithm.

- Properties of Sorting Algorithms -

(i) In-place Sorting - A sorting Algorithm which is only using fixed amount of constant space which indipendent of the size of input (array).

(ii) Stable Sort - Repeated elements appear in the same order as in the input. (i/p = o/p)

(iii) Online Sort - In the order that the input is fed to the algorithm, without having the entire input available from the start but algorithm still sort input. (data is provided to algorithm over the time).

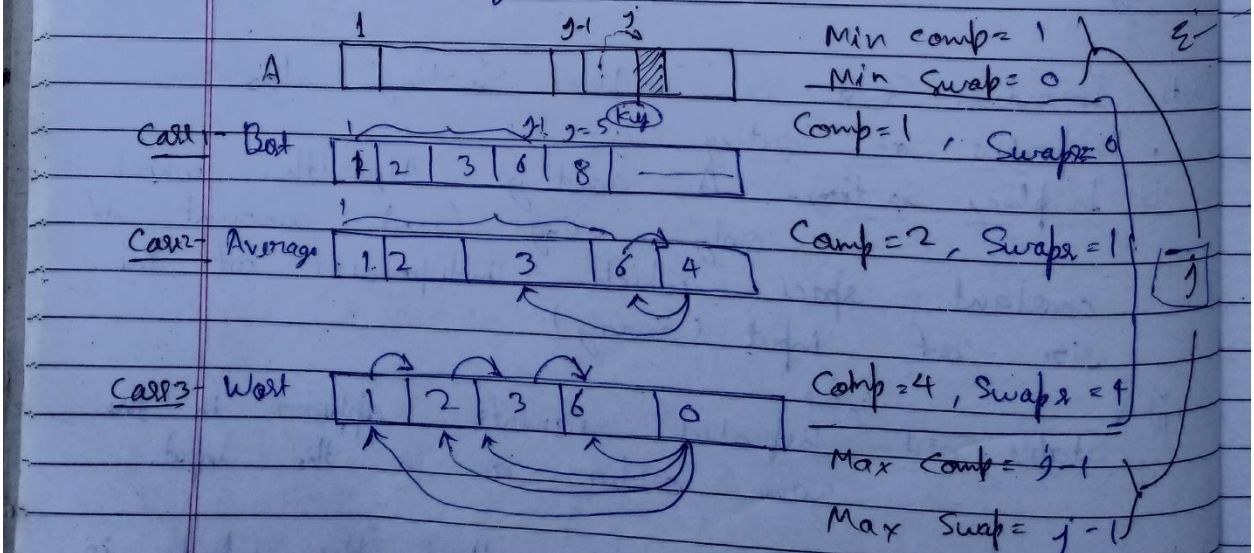
Page No.: 4
Date: 19/10/20

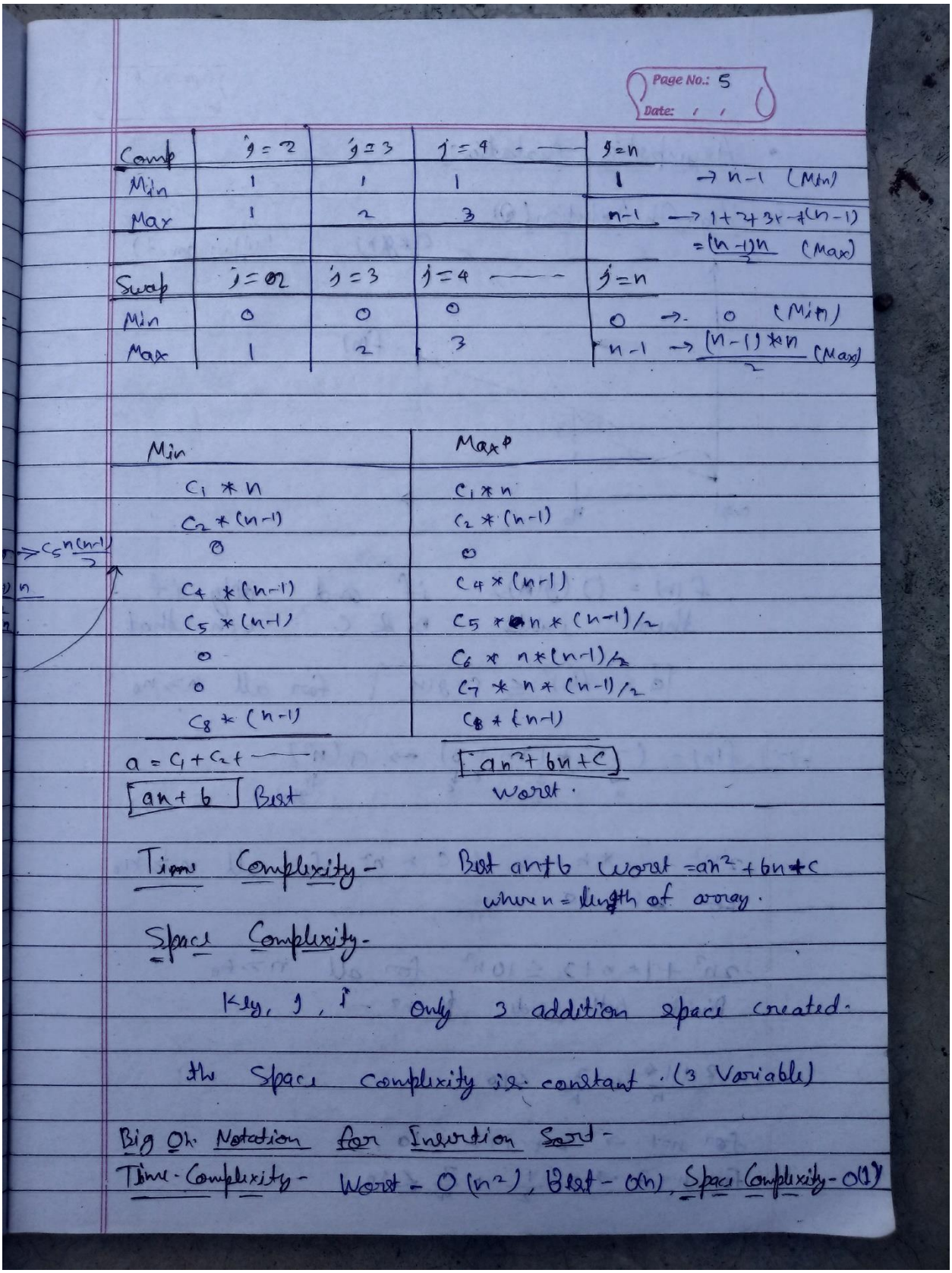
Note- Insertion Sort is "Online Sort" algorithm.

Analysing Algorithm-

Space and Time Complexity of Insertion Sort.

	Insertion Sort (A)	Times	Cost -
1)	for $j=2$ to A.length	$\rightarrow n-1$	$c_1 * n$
2)	key = A[j]	$\rightarrow n-1$	$c_2 * (n-1)$
3)	// [Insert A[j] into A[1 — j-1]]	$\rightarrow n-1$	$c_3 * (n-1)$
4)	i = j-1	$\rightarrow$	$c_4 = (n-1)$
5)	while (i>0 . And A[i] > key)	$\rightarrow$	$c_5 * (n-1)$
6)	A[i+1] = A[i]	$\rightarrow$	0 ; $c_6 (n-1)$
7)	i = i-1	$\rightarrow$	0 ; $c_7 \frac{(n-1)}{2}$
8)	A[i+1] = key.	$\rightarrow n-1$	$c_8 * (n-1)$





Page No.: 5
Date: / /

Comp	j=2	j=3	j=4 ———	j=n
Min	1	1	1	1 → n-1 (Min)
Max	1	2	3	n-1 → 1+2+3+...+(n-1) = (n-1)n/2 (Max)

Swap	j=2	j=3	j=4 ———	j=n
Min	0	0	0	0 → 0 (Min)
Max	1	2	3	n-1 → (n-1)*n/2 (Max)

Min	Max
$c_1 * n$	$c_1 * n$
$c_2 * (n-1)$	$c_2 * (n-1)$
0	0
$c_4 * (n-1)$	$c_4 * (n-1)$
$c_5 * (n-1)$	$c_5 * n * (n-1)/2$
0	$c_6 * n * (n-1)/2$
0	$c_7 * n * (n-1)/2$
$c_8 * (n-1)$	$c_8 * (n-1)$

$a = c_1 + c_2 + \cdots$

[an+b] Best

[an^2+bn+c] worst.

Time Complexity - Best an+b worst $= an^2+bn+c$
where n = length of array.

Space Complexity -

Key, i, j only 3 addition space created.

the Space complexity is constant. (3 Variable)

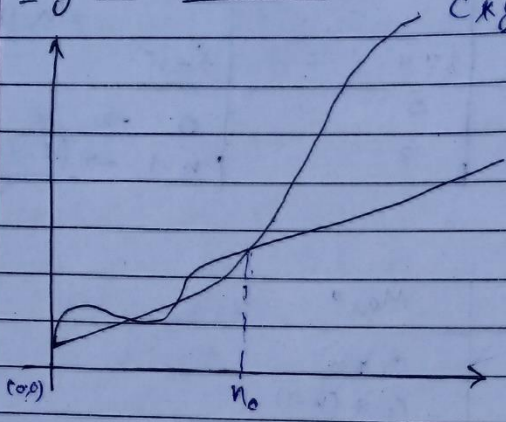
Big Oh Notation for Insertion Sort -

Time Complexity - Worst - $O(n^2)$, Best - $O(n)$, Space Complexity - $O(1)$

Page No.: 6
Date: / /

• Asymptotic Notations -

1. Big-Oh Notation (O)



$f(n) = O(g(n))$ if and only if there exists n_0 & c such that

$$0 \leq f(n) \leq c * g(n)$$ for all $n \geq n_0$

Eg- $f(n) = (2n^2 + 1*n + 3) \rightarrow O(n^2)$
(a, b, c; $g(n)$)

$2n^2 + 1*n + 3 \leq c * n^2$ for all $n \geq n_0$

Let $c = 10$

$2n^2 + 1*n + 3 \leq 10n^2$ for all $n \geq n_0$

Divide both side by n^2 -

$$2 + \frac{1}{n} + \frac{3}{n^2} \leq 10$$

for $n=1 \rightarrow 2 + 1 + 3 \leq 10$

for $n = 2 \rightarrow 2 + \frac{1}{2} + \frac{3}{4} \leq 10$

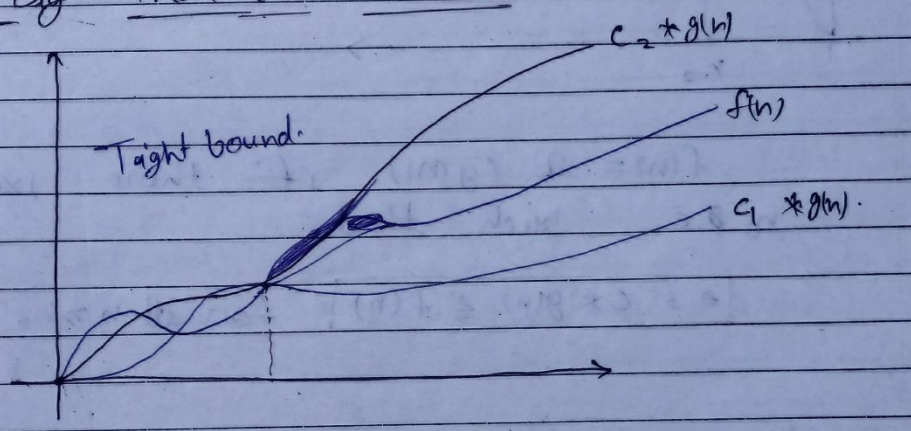
Page No.: 7
Date: / /

For $n=3 \rightarrow 2 + \frac{1}{3} + \frac{3}{9} \leq 10$

$f(n) \leq 10 * n^2$ for all $n \geq 1$

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

2- Big Theta (Θ) Notation-



$f(n) = \Theta(g(n))$ iff there exists n_0, c_1 & c_2 such that

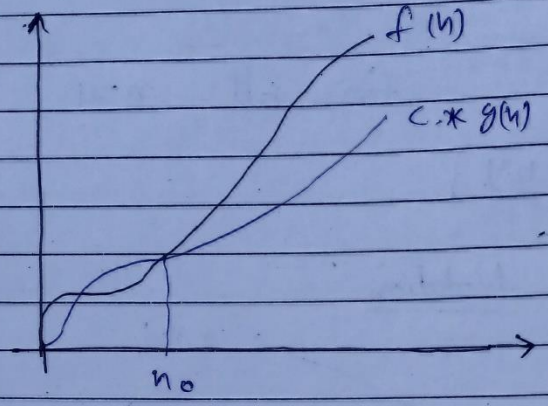
$\boxed{0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n)}$

for all $n \geq n_0$

iff - if and only if
$\forall$ - for all
$\exists$ - There exists

Page No.: 8
Date: / /

3- Big Omega (Ω) -



(Lower bound)

$f(n) = \Omega(g(n))$ iff there ~~exists~~ n_0, c, such that -

$$\boxed{0 \leq c * g(n) \leq f(n)} \quad \text{for all } n \geq n_0$$

4- Small oh (o) -

$f(n) = o(g(n))$ iff $\forall\, c > 0$, $\exists\, n_0 > 0$ such that $\boxed{0 \leq f(n) < c \cdot g(n)}$ $\forall\, n \geq n_0$

Eg- (i) $2n = O(n)$ - but $2n \neq o(n)$

for $c = 1$

$2n < 1 \cdot n$ (x)

(ii) $2n = o(n^2)$

for $c = 1$

$2 \times 1 \leq c \cdot n$ $\forall\, n \geq 2, 3, \ldots$

Page No.: 9
Date: / /

$2n = O(n)$
$2n \neq o(n)$
$2n = o(n^2)$

$2n^2 \neq o(n^2)$
$2n^2 = o(n^3)$

Alternate difinition of small oh (o) –

$f(n) = o(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$.

(i) $2n = o(n^2)$

Eg- $\lim_{n \to \infty} \frac{2n}{n^2} \to 0$

$= \lim \frac{n}{n} \to 0$

(ii) $2n = o(n)$.

$\lim_{n \to \infty} \frac{2n}{n} = 2$ Thu $2n \neq o(n)$

5- Small - Omega (ω) –

(i) $f(n) = \omega(g(n))$ iff $g(n) \prec o(f(n))$

Eg- $2n = o(n^2) \Rightarrow n^2 = \omega(n)$ $2(2n) = \omega(n)$

or

(ii) $f(n) = \omega(g(n))$ iff $\forall c > 0$ $\exists$ $n_0 > 0$ such that

$0 \leq c \cdot g(n) < f(n)$ $\forall$ $n \geq n_0$

Eg- $\frac{n^2}{2} = \omega(n)$; $\frac{n^2}{2} \neq \omega(n^2)$

Page No.: 10
Date: / /

(iii) $f(n) = \omega(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Eg- (i) $\frac{n^2}{2} = f(n)$; $g(n) = n^2$.

$$\lim_{n\to\infty} \frac{n^2/2}{n^2} = \frac{1}{2}$$

(ii) $g(n) = n$ $\lim_{n\to\infty} \frac{n^2/2}{n} = \infty$

Intuition of Notation-

$f(n) = O(g(n)) \longrightarrow f \leq g$
$f(n) = \Omega(g(n)) \longrightarrow f \geq g$
$f(n) = \Theta(g(n)) \longrightarrow f = g$
$f(n) = o(g(n)) \longrightarrow f < g$
$f(n) = \omega(g(n)) \longrightarrow f > g$

Relationships between various notations-

1- Transitive Relationship- It hold for all notations $O, \Theta, \Omega, o, \omega$

Eg- (i) If $f(n) = \Theta(g(n))$ AND $g(n) = \Theta(h(n))$
then $f(n) = \Theta(h(n))$

Using intuition-
If $f = g$ & $g = h$ then $f = h$

(ii) If $f(n) = \omega(g(n))$ AND $\omega(h(n))$
then $f(n) = \omega(h(n))$

$f > g$ & $g > h$ then $f > h$.

Page No.: 11
Date: / /

2- Reflexive Relationship- It hold for the notations O, Θ and Ω

$f(n) = \Theta(f(n)) \longrightarrow f = f$ ✓
$f(n) = O(f(n)) \longrightarrow f \leq f$ ✓
$f(n) = \Omega(f(n)) \longrightarrow f \geq f$ ✓
$f(n) \neq o(f(n)) \longrightarrow f < f$ ✗
$f(n) \neq \omega(f(n)) \longrightarrow f > f$ ✗

3- Symmetric Relationship- It hold for Θ notation only.

Eg- $f(n) = \Theta g(n)$ iff $g(n) = \Theta(f(n))$
$f = g$ iff $g = f$.

~~4- Transitive~~

4- Transpose Symmetry-

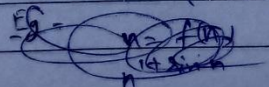
i) $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$
Eg- $f \leq g$ iff $g \geq f$

ii) $f(n) = o(g(n))$ iff $g(n) = \omega(f(n))$
$f < g$ iff $g > f$

5- Trichotomy- Not hold for any notation-
a, b, then $a > b$; $a < b$; $a = b$ (one of these true always.)

$f(n) = O(g(n))$
$f(n) = \omega(g(n))$
$f(n) = \Theta(g(n))$

Eg-



Page No.: 12
Date: / /

Eg- $n = f(n)$
$n^{1+\sin n} = g(n)$

$f(n)$
↓
n.

$n^{1+\sin(n)}$
$n^{1+(-1)} \leftrightarrow n^{1+1}$
$n^0 \leftrightarrow n^2$
$1 \leftrightarrow n^2$

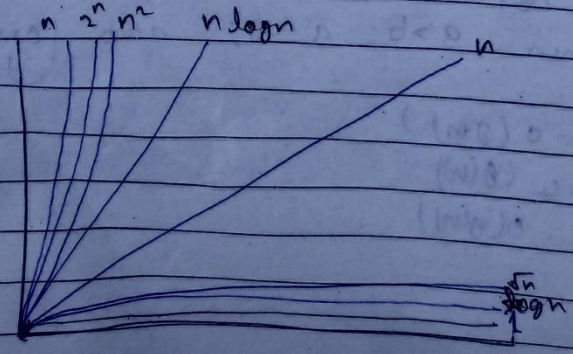
We can't compare the $f(n)$ & $g(n)$

Order of Common functions & real word applications-

Order of function-

n	n^2	$n \log_{10} n$	2^n
1	1	$1 \times 0 = 0$	2
10	100	$10 \times 1 = 10$	$2^{10} = 1024$
100	10^4	$100 \times 2 = 200$	$2^{100} = 1.28 \times 10^{30}$
1000	10^6	$1000 \times 3 = 3000$	$2^{100} = 1.6 \times 10^{301}$
10^4	10^8	4×10^4	Very Very large
10^5	10^{10}	5×10^5	Very Very large.

$1 < \log n < \sqrt{n} < n \log n < n^2 < 2^n < n!$



Page No.: 13
Date: / /

Notation	Name
$O(1)$	Constant
$O(\log^* n)$	log * Star
$O(\log n)$	logarithmic
$O((\log n)^c)$ $c>1$	Polylogarithmic
$O(n^c)$ $0<c<1$	Fractional Power
$O(n)$	Linear
$O(n \log^* n)$	n log starn
$O(n \log n) = O(\log n!)$	linearithmic, loglinear, quasilinear
$O(n^2)$	quadratic
$O(n^c)$	Polynomial or algebraic
$O(c^n)$ $c>1$	exponential
$O(n!)$	Factorial

Page No.: 14
Date: 20/10/20

Q- $f(n) = n^2+n+1$ what are the valid functions for $g(n)$, $h(n)$ & $k(n)$?

(i) $f(n) = o(g(n))$ $g(n) = n^2 ; n^3, n^4, n^5 \ldots$

(ii) $f(n) = \Omega(h(n))$ $h(n) = n^2, n$

(iii) $f(n) = \Theta(k(n))$ $k(n) = n^2$

(i) $f(n) = n^2+n+1$

Let $g(n) = n^2$, $n \geq 0$

$0 \leq f(n) \leq c\, g(n)$ $\forall\ n \geq n_0$, $\exists\ c, n_0$

$n^2 \leq n^2$

$n^2+n+1 \leq n^2+n^2+n^2$ $\forall\ n \geq 1$

$n^2+n+1 \leq 3n^2$ $\forall\ n \geq 1$

(c = 3, n_0 = 1)

$f(n) = o(n^2)$, $g(n) = n^2$

Let $g(n) = n^3$, $f(n) = n^2+n+1$

$n^2 \leq n^3$ $\forall\ n \geq 1$

$n^2+n+1 \leq n^3+n^3+n^3$ $\forall\ n \geq 1$

$n^2+n+1 \leq 3n^3$ $\forall\ n \geq 1$

$g(n) = n^3$

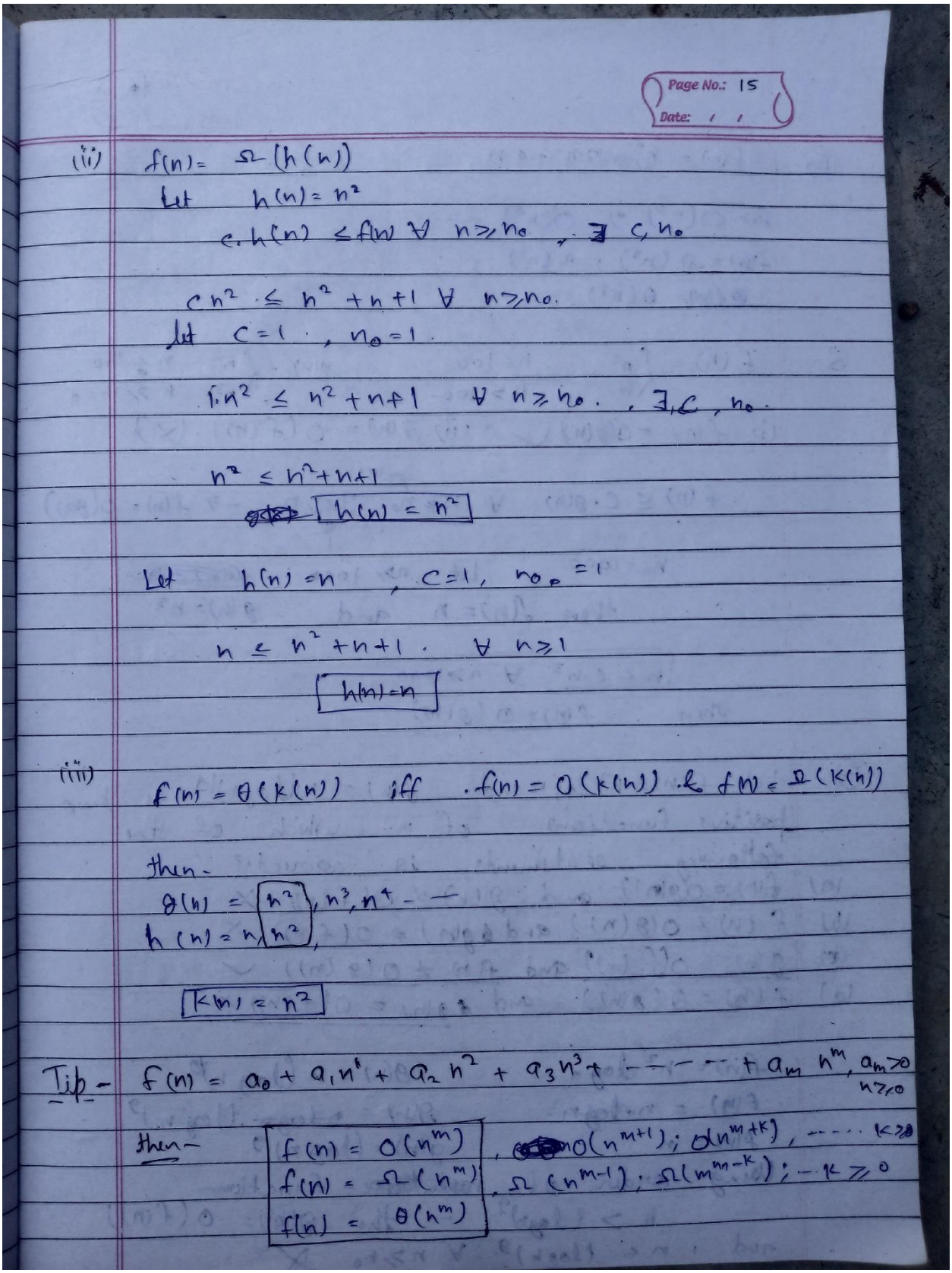
Let $g(n) = n$, $f(n) = n^2+n+1$

$n^2+n+1 \leq cn$ $\forall\ n \geq n_0$.

divide by n on both side.-

$n+1+\frac{1}{n} \leq c$ ✗

$g(n) \neq n$



Page No.: 15
Date: / /

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

Let $h(n) = n^2$

$\therefore h(n) \leq f(n) \quad \forall \quad n \geq n_0 \quad , \exists \; c, n_0$

$cn^2 \leq n^2 + n + 1 \quad \forall \quad n \geq n_0.$

Let $c = 1 \; , \; n_0 = 1.$

$\therefore n^2 \leq n^2 + n + 1 \quad \forall \quad n \geq n_0. \; , \exists \; c, n_0.$

$n^2 \leq n^2 + n + 1$

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

Let $h(n) = n \quad , c = 1, \; n_0 = 1$

$n \leq n^2 + n + 1. \quad \forall \quad n \geq 1$

$\boxed{h(n) = n}$

(iii) $f(n) = \theta(k(n))$ iff $f(n) = O(k(n))$ & $f(n) = \Omega(k(n))$

then -

$\theta(n) = \boxed{n^2}, n^3, n^4 - -$

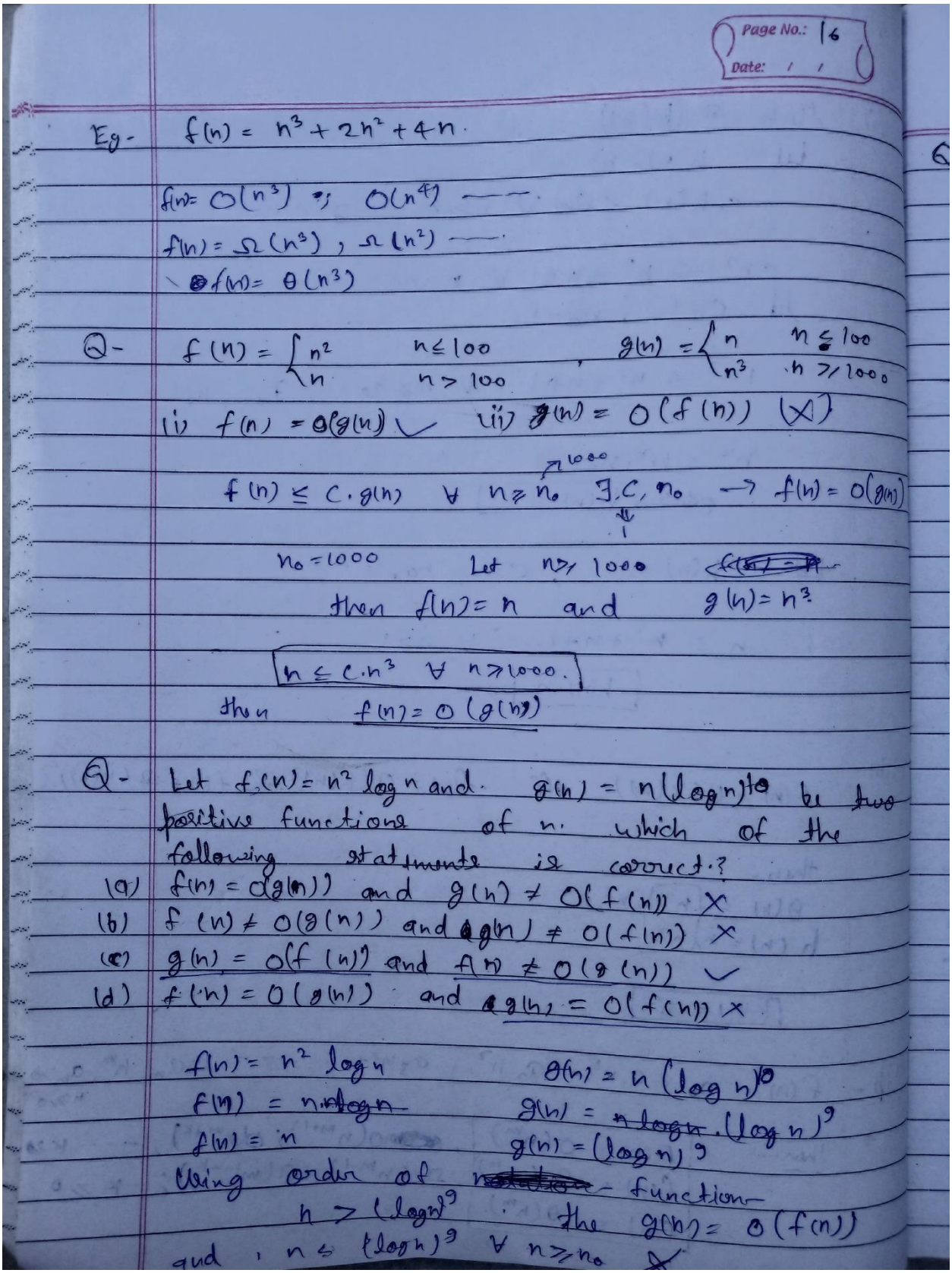
$h(n) = n, \boxed{n^2},$

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

Tip - $f(n) = a_0 + a_1 n^1 + a_2 n^2 + a_3 n^3 + - - - - - + a_m n^m, \; a_m > 0, \; n > 0$

then - $\boxed{\begin{array}{l} f(n) = O(n^m) \\ f(n) = \Omega(n^m) \\ f(n) = \theta(n^m) \end{array}}$, $O(n^{m+1}); O(n^{m+k}), - - - - \; k \geq 0$

$\Omega(n^{m-1}); \Omega(n^{m-k}); - k \geq 0$



Page No.: 16
Date: / /

Eg- $f(n) = n^3 + 2n^2 + 4n$.

$f(n) = O(n^3)$; $O(n^4)$ ——

$f(n) = \Omega(n^3)$, $\Omega(n^2)$ ——

$\therefore f(n) = \Theta(n^3)$

Q- $f(n) = \begin{cases} n^2 & n \le 100 \\ n & n > 100 \end{cases}$, $g(n) = \begin{cases} n & n \le 100 \\ n^3 & n \ge 1000 \end{cases}$

(i) $f(n) = O(g(n))$ ✓ (ii) $g(n) = O(f(n))$ (X)

$f(n) \le c \cdot g(n)$ $\forall$ $n \ge n_0$ $\exists, c, n_0 \rightarrow f(n) = O(g(n))$

$n_0 = 1000$ Let $n \ge 1000$

then $f(n) = n$ and $g(n) = n^3$

$n \le c \cdot n^3$ $\forall$ $n \ge 1000$.

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

Q- Let $f_1(n) = n^2 \log n$ and $g(n) = n(\log n)^{10}$ be two positive functions of n. which of the following statements is correct.?

(a) $f(n) = O(g(n))$ and $g(n) \ne O(f(n))$ X

(b) $f(n) \ne O(g(n))$ and $g(n) \ne O(f(n))$ X

(c) $g(n) = O(f(n))$ and $f(n) \ne O(g(n))$ ✓

(d) $f(n) = O(g(n))$ and $g(n) = O(f(n))$ X

$f(n) = n^2 \log n$ $g(n) = n(\log n)^{10}$

$f(n) = n \cdot \log n$ $g(n) = n \log n \cdot (\log n)^9$

$f(n) = n$ $g(n) = (\log n)^9$

Using order of function

$n > (\log n)^9$. the $g(n) = O(f(n))$

and , $n \le (\log n)^9$ $\forall$ $n \ge n_0$ ✗

Page No.: 17
Date: / /

Q- $f_1(n) = 2^n$; $f_2(n) = n^{3/2}$; $f_3(n) = n\log_2 n$; $f_4 = n^{\log_2 n}$
What is the increasing order of asymptotic complexity?

Order of functions -

$f_3 = n\log n$	$f_2 = n^{3/2}$
$= n/\log n$	$= n \cdot n^{1/2}$
$\log n$	$n^{1/2}$

$f_3 < f_2$

$f_2 = n^{3/2}$	$f_4 = n^{\log_2 n}$
$\frac{3}{2}$	$\log_2 n$

$n^{3/2} < n^{\log_2 n}$

$f_2 < f_4$

$f_1 = 2^n$ $\quad f_4 = n^{\log_2 n}$

take log of both side

$n\log_2 2$	$\log(n^{\log_2 n})$
n	$\log_2 n \log_2(n)$
n	$(\log_2 n)^2$

$f_1 > f_4$

$f_3 < f_2 < f_4 < f_1$

Page No.: 18
Date: / /

Q- In a permutation $a_i \dots a_n$ of n distinct integers, an inversion is a pair (a_i, a_j) such that $i < j$ and $a_i > a_j$.
What would be the worst case time complexity of the Insertion Sort Algorithm, if the inputs are related to permutations of 1 ... n with at most n inversions?

(i) $\Theta(n^2)$
(ii) $\Theta(n \log n)$
(iii) $\Theta(n^{1.5})$
(iv) $\Theta(n)$ ✓

Eg- | 1 | 2 | 3 | 4 | 5 |

↓ inversion → swap (move to left)

(i) 2, 1, 3, 4, 5 → (2, 1)

(ii) 2, 3, 1, 4, 5 → (2, 1) (3, 1)

(iii) 2, 3, 4, 5, 1 → (4, 1) (3, 1), (2, 1) (5, 1).

Worst Case Complexity - $O(n+m)$
↳ inversions in my array

in question given
there at most n inversion
$n = m$

$O(n+m) = O(2n) = \Theta(2n) = \Theta(n)$

Page No.: 19
Date: 22/10/20

Q.1- Which are True or False.

(i) $(n+k)^m = O(n^m)$ (True)

using binomial expansion -

$(n+m)^m = n^m + {}^mC_1 n^{m-1} k^1 + {}^mC_2 n^{m-2} k^2 + \cdots + k^m n^0$

$= n^m + c_1 n^{m-1} + c_2 n^{m-2} + \cdots + c_m n^0$

Here k & m are constant then we write as c. & we known maximum power of polynomial is Order of function.

$= O(n^m)$

So. $\boxed{(n+k)^m = O(n^m)}$

(ii) $2^{n+1} = O(2^n)$ (True.)

$2 \cdot 2^n \leq c \cdot 2^n \quad \exists\, c, n_0 \quad \forall\ n \geq n_0$

let $n_0 = 1$ & $c = 3$

then

$2 \cdot 2^n \leq 3 \cdot 2^n$

$2 \leq 3$. (True.)

(iii) $2^{2n+1} = O(2^n)$. (False)

$2^{2n+1} \leq c \cdot 2^n \quad \exists\, c, n_0, \quad \forall\ n \geq n_0$

Let $c = 2$.

$2 \cdot 2^{2n} \leq 2^n$

$2^{2n} \leq 2^n$

$\log 2^{2n} \leq \log 2^n$

$2n \log 2 \leq n \log 2$

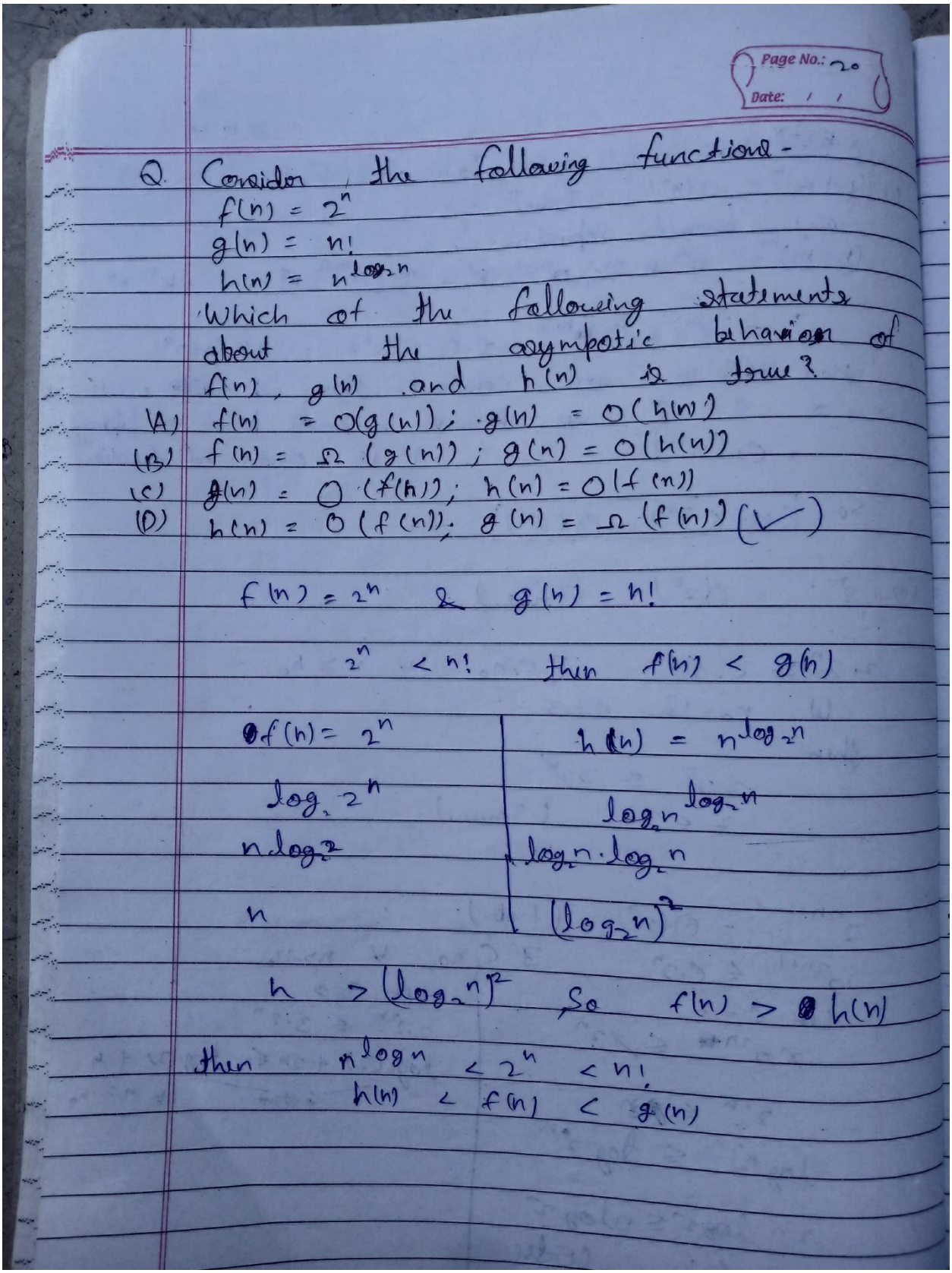
$2n \leq n$. (False

Let $c = 3$

$2 \cdot 2^{2n} \leq 3 \cdot 2^n$

$\log(2) + 2n \leq \log(3) + n$

(Constant) $\forall\ n \geq n_0$



Page No.: 20
Date: / /

Q. Consider the following functions -
$f(n) = 2^n$
$g(n) = n!$
$h(n) = n^{\log_2 n}$
Which of the following statements about the asymptotic behaviour of $f(n)$, $g(n)$ and $h(n)$ is true?

(A) $f(n) = O(g(n))$; $g(n) = O(h(n))$
(B) $f(n) = \Omega(g(n))$; $g(n) = O(h(n))$
(C) $g(n) = O(f(n))$; $h(n) = O(f(n))$
(D) $h(n) = O(f(n))$; $g(n) = \Omega(f(n))$ (✓)

$f(n) = 2^n$ & $g(n) = n!$

$2^n < n!$ then $f(n) < g(n)$

$f(n) = 2^n$	$h(n) = n^{\log_2 n}$
$\log_2 2^n$	$\log_2 n^{\log_2 n}$
$n \log_2 2$	$\log_2 n \cdot \log_2 n$
n	$(\log_2 n)^2$

$n > (\log_2 n)^2$ So $f(n) > h(n)$

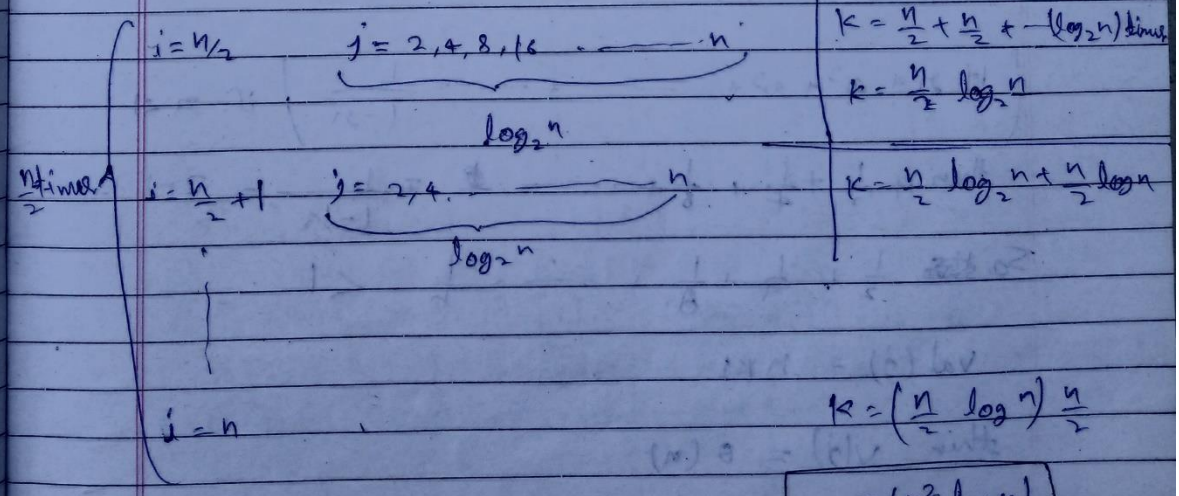
then $n^{\log n} < 2^n < n!$
$h(n) < f(n) < g(n)$

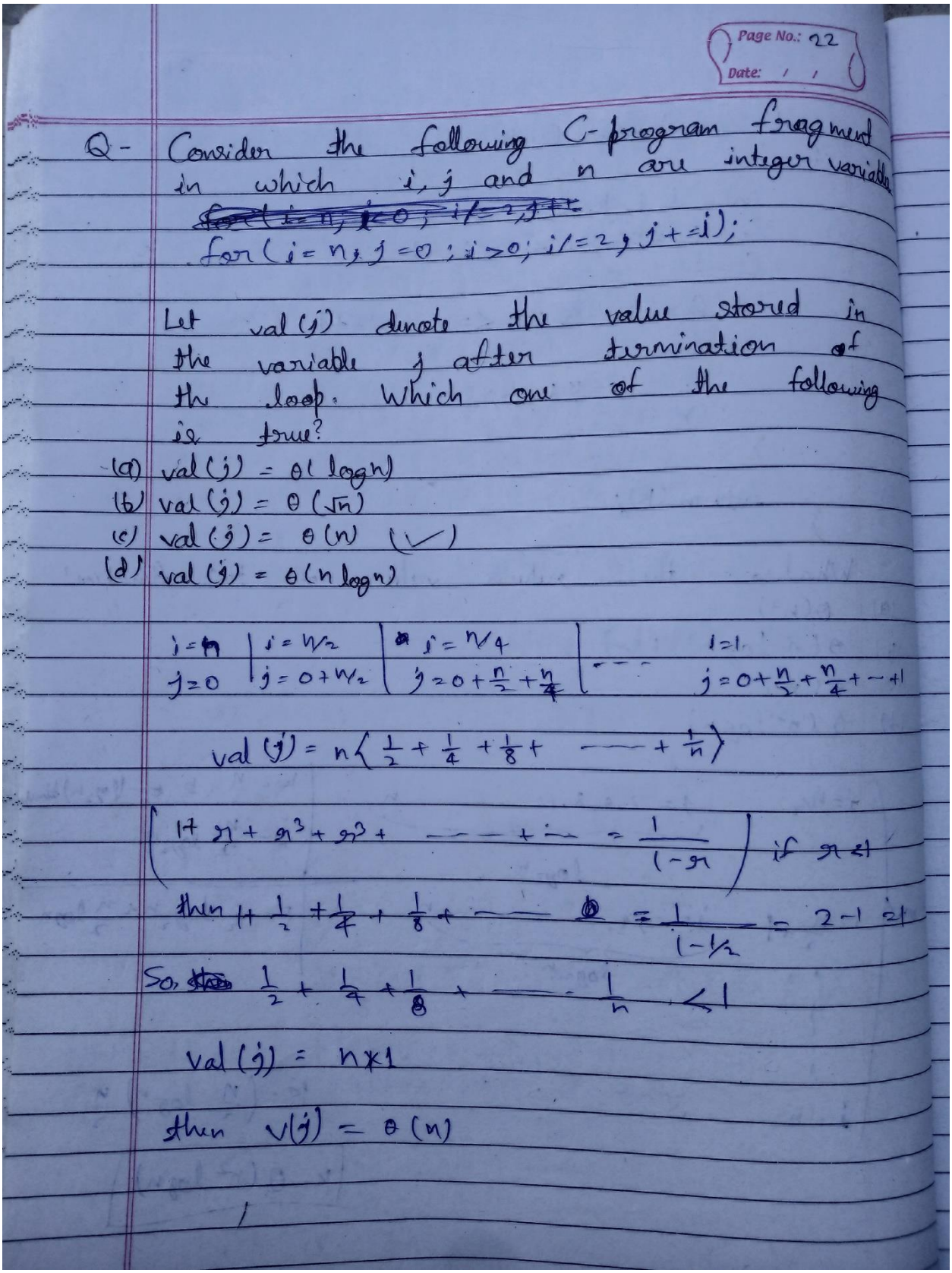
Page No.: 21
Date: / /

Q-
```
int unknown (int n)
{
    int i, j, k = 0;
    for (i = n/2; i <= n; i++)
    {
        for (j = n/2; j <= n; j++)
        {
            k = k + n/2;
        }
    }
    return (k);
}
```

What is the return value of this function?

(a) $\Theta(n^2)$
(b) $\Theta(n^2 \log n)$ (✓)
(c) $\Theta(n^3)$
(d) $\Theta(n^3 \log n)$





Page No.: 22
Date: / /

Q- Consider the following C-program fragment in which i, j and n are integer variables.

~~for(i=n, j=0; i/=2, j++;~~

for(i=n; j=0; i>0; i/=2, j+=i);

Let val(j) denote the value stored in the variable j after termination of the loop. Which one of the following is true?

(a) val(j) = θ(log n)
(b) val(j) = θ(√n)
(c) val(j) = θ(n) (✓)
(d) val(j) = θ(n log n)

i = n	i = n/2	i = n/4	...	i = 1
j = 0	j = 0 + n/2	j = 0 + n/2 + n/4	...	j = 0 + n/2 + n/4 + ... + 1

$$val(j) = n\left\{\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{n}\right\}$$

$$\left(1 + r + r^2 + r^3 + \cdots + \infty = \frac{1}{(1-r)}\right) \text{ if } r < 1$$

then $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \infty = \frac{1}{1-1/2} = 2 > 1$

So, $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \frac{1}{n} < 1$

val(j) = n × 1

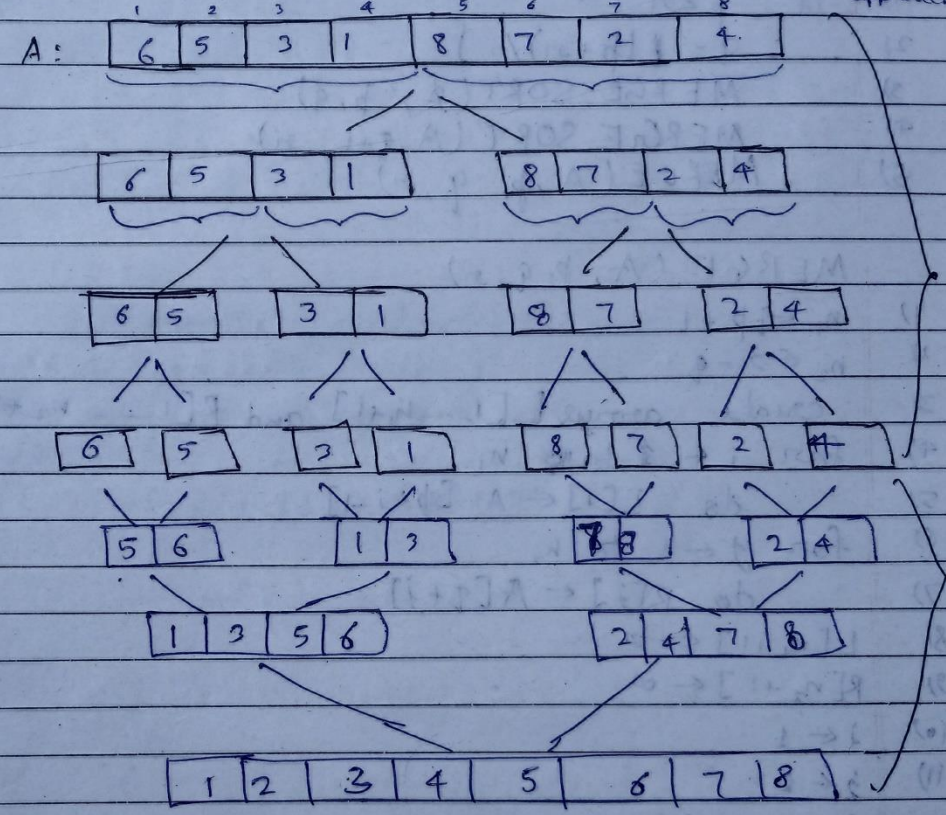
then v(j) = θ(n)

Page No.: 23
Date: 23/10/20

Merge Sort -

→ Intution- It is based on divide & conquere approach.

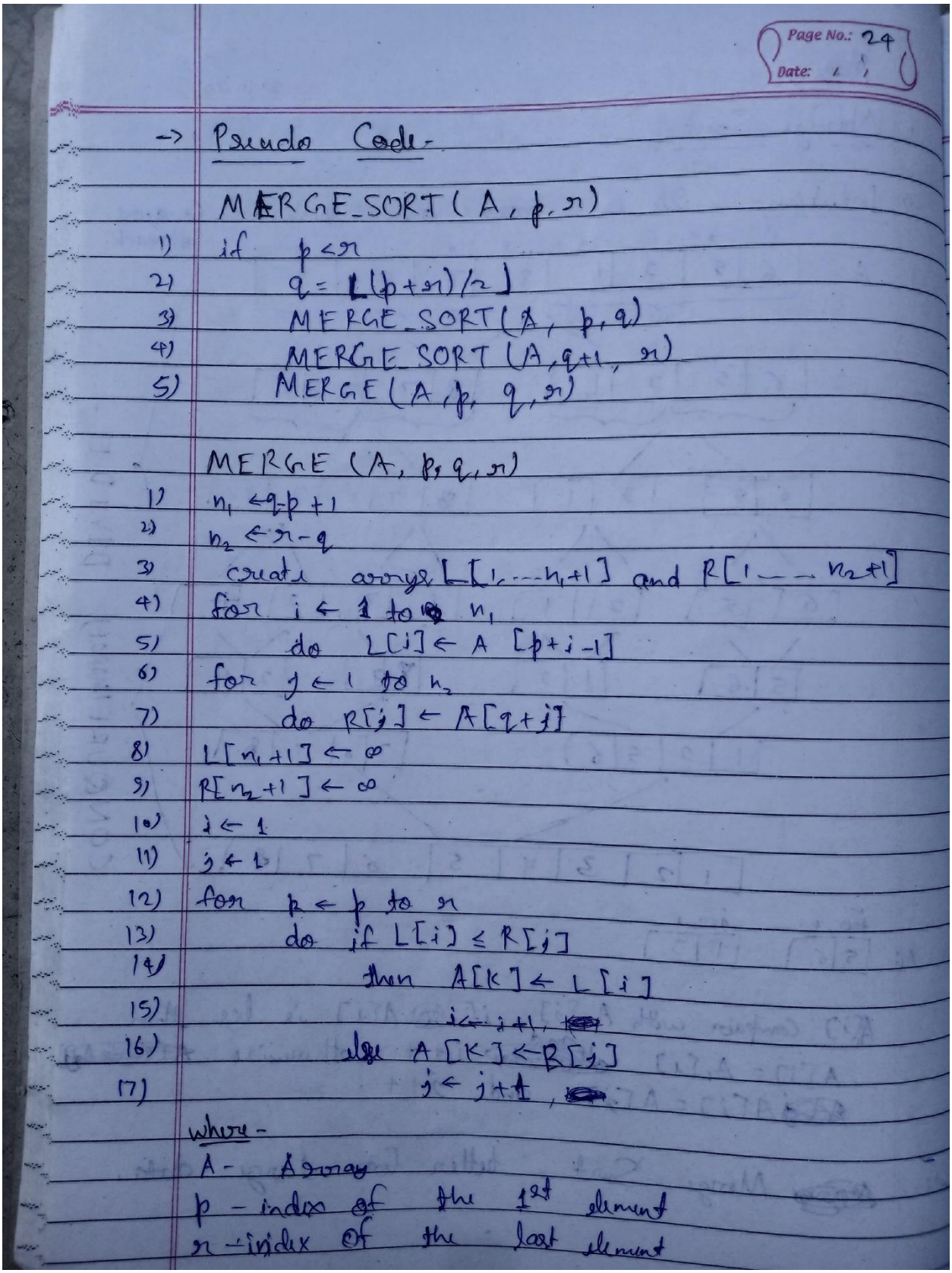
Eg A: 6 5 3 1 8 7 2 4



A: 5 6 | 1 3 (i=0, 1 ; j=0, 1)

A[i] compair with $A_2[i]$ if A[i] is less than $A_1[i]$ and i++ otherwise $A[i] = A_2[i]$ and j++.

Note- Merge Sort better for large data.


Page No.: 24
Date:
-> Pseudo Code-
MERGE_SORT (A, p, r)
1) if p < r
2) q = ⌊(p+r)/2⌋
3) MERGE_SORT (A, p, q)
4) MERGE_SORT (A, q+1, r)
5) MERGE (A, p, q, r)
MERGE (A, p, q, r)
1) n1 ← q-p+1
2) n2 ← r-q
3) create arrays L[1 ... n1+1] and R[1 ... n2+1]
4) for i ← 1 to n1
5) do L[i] ← A[p+i-1]
6) for j ← 1 to n2
7) do R[j] ← A[q+j]
8) L[n1+1] ← ∞
9) R[n2+1] ← ∞
10) i ← 1
11) j ← 1
12) for k ← p to r
13) do if L[i] ≤ R[j]
14) then A[k] ← L[i]
15) i ← i+1,
16) else A[k] ← R[j]
17) j ← j+1,
where -
A - Array
p - index of the 1st element
r - index of the last element

Page No.: 25
Date: 24/09/20

→ Code -

```
void merge(int arr[], int lb, int mid, int ub)
{
    int n1 = mid - lb + 1;
    int n2 = ub - mid;
    int left[n1], right[n2];
    for(int i=0; i<n1; i++)
        left[i] = arr[lb+i];
    for(int j=0; j<n2; j++)
        right[j] = arr[mid+1+j];
    int i=0, j=0, k=lb;
    while(i<n1 && j<n2)
    {
        if(left[i] <= right[j])
        {
            arr[k] = left[i];
            i++;
        }
        else
        {
            arr[k] = right[j];
            j++;
        }
        k++;
    }
    while(i<n1)
    {
        arr[k] = left[i];
        i++;
        k++;
    }
```

Page No.: 26
Date: / /

```
    while (j<n2)
    {
        arr[k] = right[j];
        j++;
        k++;
    }
}
void mergeShot(int arr[], int lb, int ub)
{
    if (lb < ub)
    {
        int mid = (lb + ub)/2;   lb + (ub-lb)/2;
        mergeSort(arr, lb, mid);
        mergeSort(arr, mid+1, ub);
        merge(arr, lb, mid, ub);
    }
}
```

→ Analyzing time & Space Complexity -

Time Complexity.

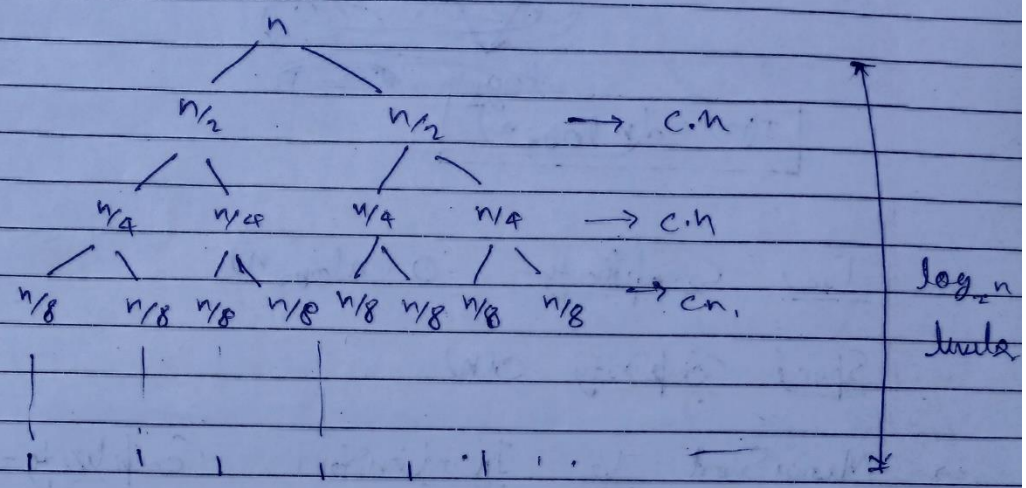
$$T(n) = T(n/2) + T(n/2) + O(n)$$
$$T(n) = 2 * T(n/2) + O(n)$$

Space Complexity -

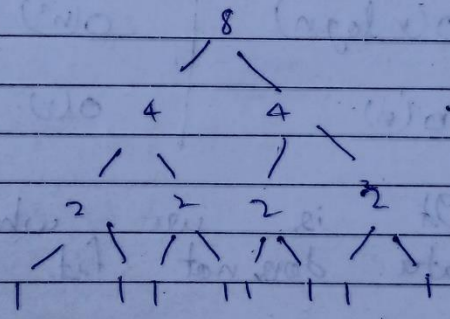
$$S(n) = C + O(n)$$

Page No.: 27
Date: / /

→ Recursion Tree method - (Intution) -



for n = 8



n = 8 → 3 levels → 3 . c.n
n = 16 → 4 levels → 4 . c.n
n = 32 → → 5 levels → 5 . c.n.

$\log_2 n = x$

$2^x = n$.

n = 8 no. levels. $2^x = 8$.

$x = 3$

Page No.: 28
Date: / /

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

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

→ Time Complexity- $O(n \log_2 n)$

Space Complexity $O(n)$

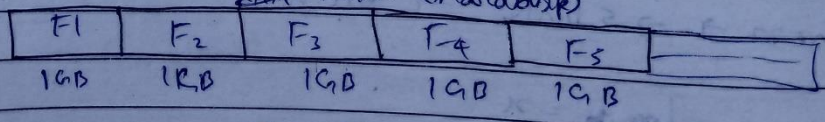
→ Merge Sort Vs Insertion Sort Complexity-

	Merge Sort	Insertion Sort
Time → Worst →	$O(n \log n)$	$O(n)$
Time → Best →	$O(n \log n)$	$O(n^2)$
Space →	$O(n)$	$O(1)$.

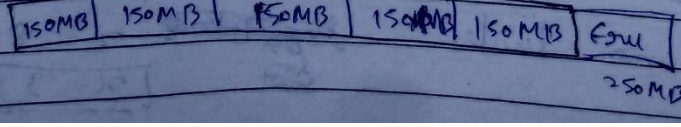
→ External Sort (Merge)- It is use when the data does not fit into RAM.

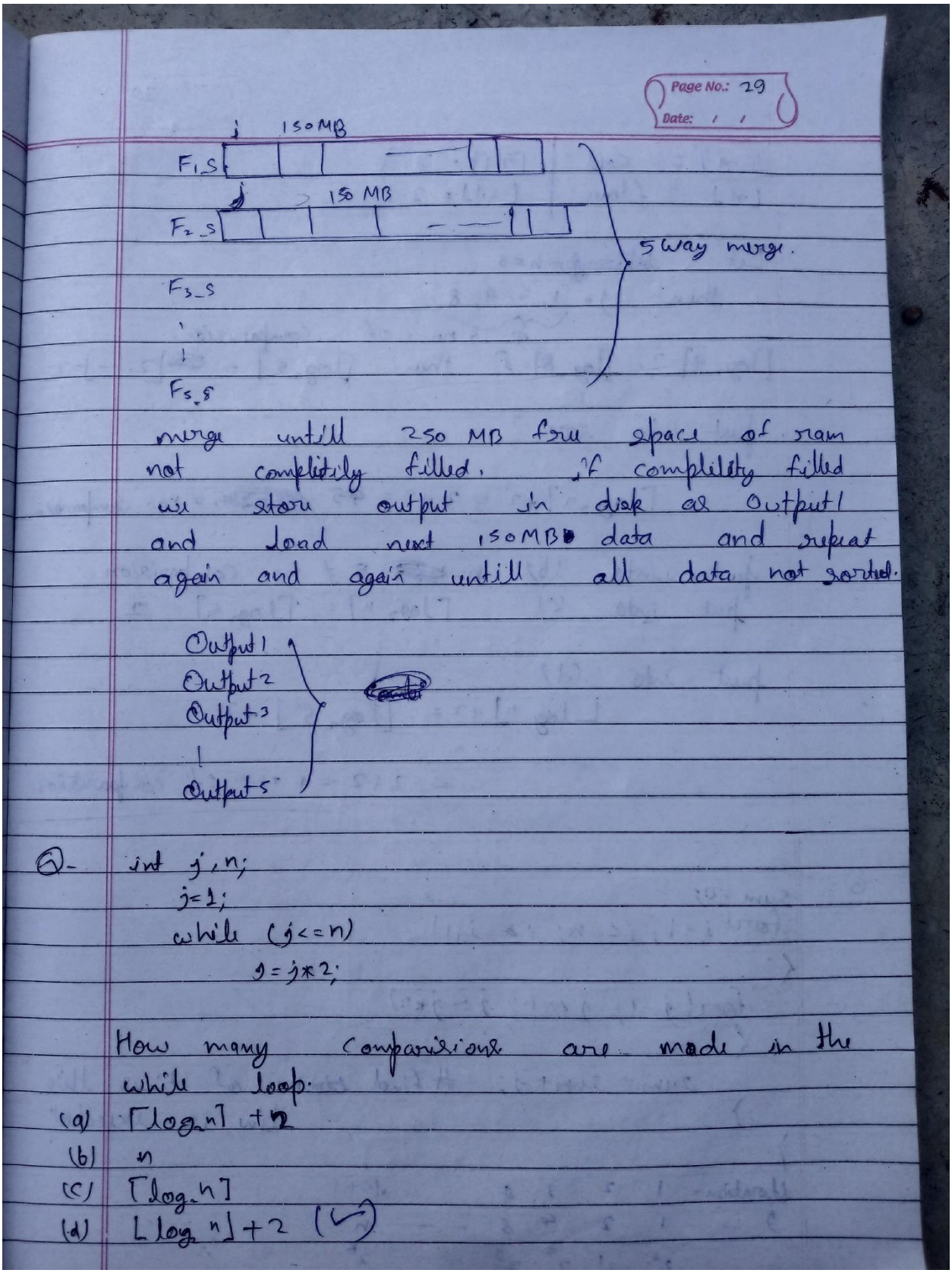
It use k way merge approach.

Eg You have 1 GB Ram and Sort 5 GB data



we load 150 MB of all slots.




Page No.: 29
Date: / /
150MB
F1_S
150 MB
F2_S
5 way merge.
F3_S
F5_S
merge untill 250 MB free space of ram not completely filled. If completely filled we store output in disk as Output1 and load next 150MB data and repeat again and again untill all data not sorted.
Output1
Output2
Output3
Output5
Q- int j, n;
j=1;
while (j<=n)
j=j*2;
How many comparisions are made in the while loop.
(a) ⌈log2n⌉ +2
(b) n
(c) ⌈log2n⌉
(d) ⌊log2 n⌋+2 (✓)

Page No.: 30
Date: / /

$\lceil x \rceil$ = ceil | $\lceil 2.1 \rceil = 3$
$\lfloor x \rfloor$ = floor | $\lfloor 2.1 \rfloor = 2$

Let $n = 5$

then $j = 1, 2, 4, 8$ → no. of comparision.

$\lceil \log_2 4 \rceil = 2$, $\lceil \log_2 8 \rceil = 3$ then $\lceil \log_2 5 \rceil = \lfloor 2.- \rfloor = 2$

put into (a).

$\lceil \log_2 n \rceil + 2 = 3 + 2 = 5 \neq$ no. comparisons

put into (b) $n = 5 \neq$ no. comparision.

put into (c) $\lceil \log_2 n \rceil = \lceil \log_2 5 \rceil = 2$

put into (d)

$\lfloor \log n \rfloor + 2 = \lfloor \log_2 5 \rfloor + 2$

$= 2 + 2 = 4 =$ no. of comparision

Q-

```
sum = 0;
for(i = 1; i <= n; i = i+1)
{
    for(j = 1; j <= n; j = j*2)
    {
        sum = sum + j;   # find no. of time this line is executed
    }
}
```

iteration -	1	2	3	4	———	$i = i+1$
j -	1	2	4	8	———	n
	2^0	2^1	2^2	2^3	———	2^k

Page No.: 31
Date: / /

i → 1 , 2 ——— n
j → O(log n) , O(log n) ——— O(log n)

n times

then O(n * log n)

Q- Assume that a mergesort algorithm in the worst case takes 30 seconds for an input of size 64. Which of the following most closely approximates the maximum input size of a problem that can be solved in 6 minutes?

(A) 256
(B) 512
(C) 1024
(D) 2018

given- n = 64, and n log n = 30 sec.

$c \cdot n \log n = 30$ if $n = 64$

$c * 64 \log_2 64 = 30$

$c * 64 * 6 = 30$

$c = \frac{30}{(64 * 6)}$

$c * m \log m = 360$ (6 minutes = 360 sec

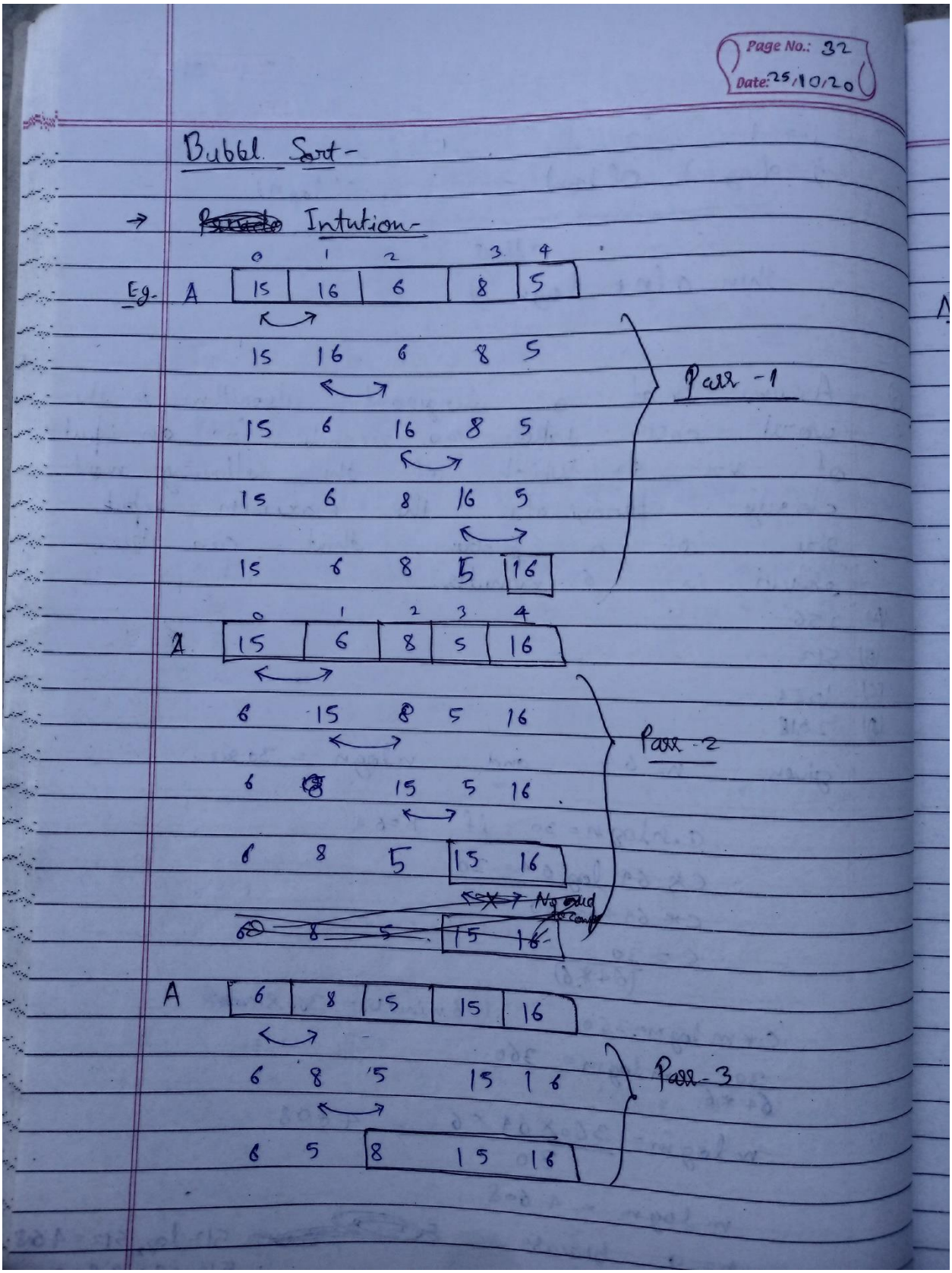
$\frac{30}{64 * 6} m \log m = 360$

$m \log m = \frac{360 \times 64 \times 6}{30} = 4608$

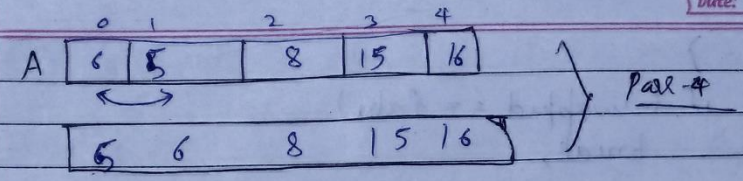
$m \log m = 4608$

$m = 9$, because $512 \log_2 512 = 4608$

$= 512 \times 9 = 4608.$


Page No.: 32
Date: 25/10/20
Bubble Sort-
→ Intution-
Eg. A 15 16 6 8 5
Pass -1
Pass -2
No swap
Pass-3

Page No.: 33
Date: / /



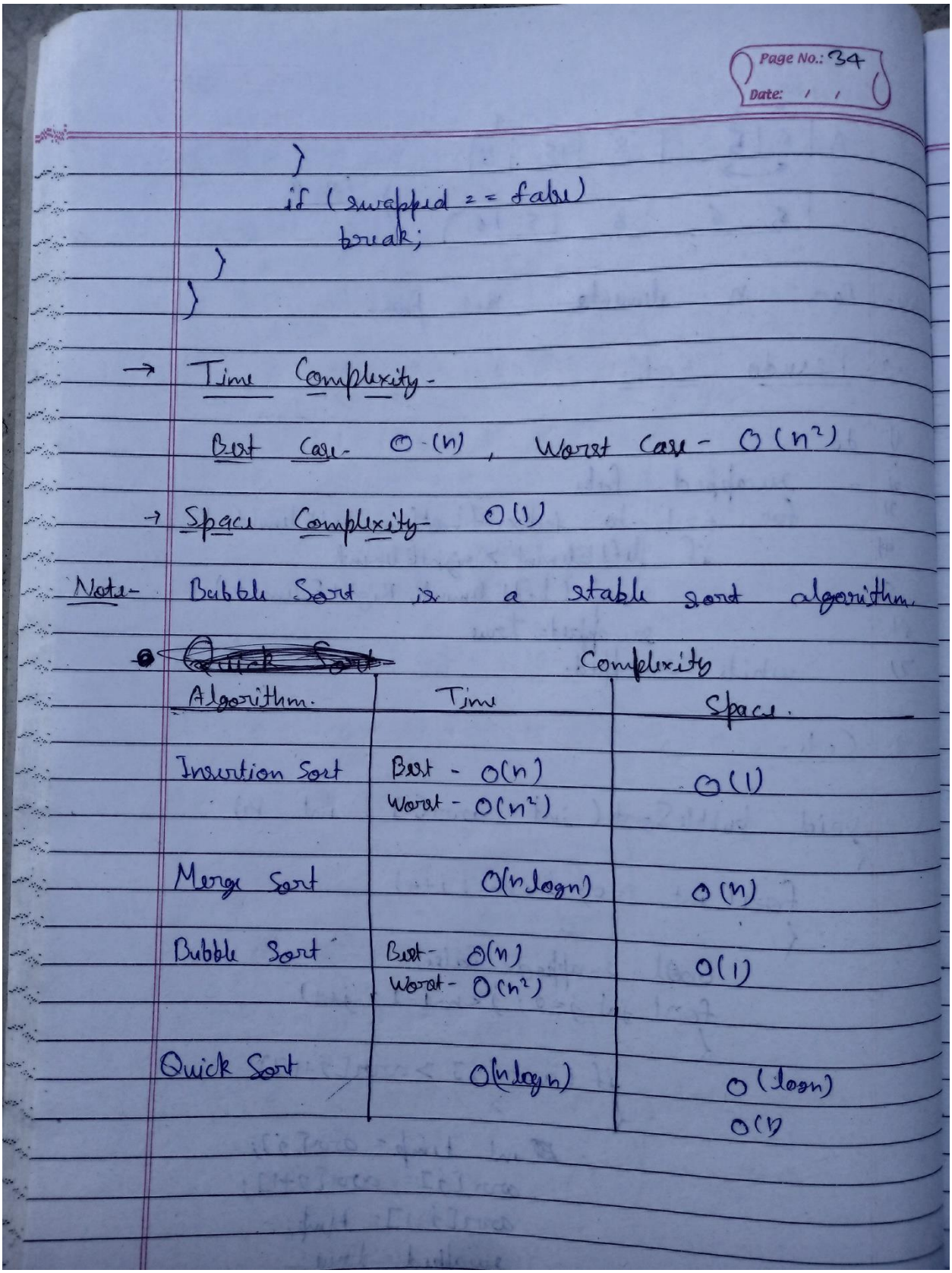
Note- for n elements n-1 pass.

→ Pseudo Code-

```
1) do
2)     swapped = false
3)     for i = 1 to index of Last Unsorted Element -1
4)         if leftElement > rightElement
5)             swap (leftElement, RightElement)
6)             swapped = true
7)     while swapped.
```

→ Code-

```
void bubbleSort( int arr[], int n)
{
    for(int i=0; i<n-1; i++)
    {
        bool swapped= false;
        for( int j=0; j<n-i-1; j++)
        {
            if (arr[j] > arr[j+1])
            {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
                swapped = true;
            }
```



Page No.: 34
Date: / /

```
        }
        if (swapped == false)
            break;
    }
}
```

→ Time Complexity -

Best Case - $O(n)$, Worst Case - $O(n^2)$

→ Space Complexity - $O(1)$

Note - Bubble Sort is a stable sort algorithm.

Complexity

Algorithm.	Time	Space.
Insertion Sort	Best - $O(n)$ Worst - $O(n^2)$	$O(1)$
Merge Sort	$O(n \log n)$	$O(n)$
Bubble Sort	Best - $O(n)$ Worst - $O(n^2)$	$O(1)$
Quick Sort	$O(n \log n)$	$O(\log n)$ $O(1)$

Page No.: 35
Date: / /

Quick Sort - (General Purpose sorting algorithm)

→ Intution-

Eg- A.

0	1	2	3	4	5	6	7	8
7	6	10	5	9	2	1	15	7

start start start end

Pivot = 7

A[0] $7 \le 7$, $6 \le 7$, $10 \le 7$ (X) stop start.

stop < end swap start and end

7	6	7	5	9	2	1	15	10

start end end end

$7 \le 7$, $5 \le 7$, $9 \le 7$ (X) stop start

start < end (✓) decrese end.

$10 \ge 7$; $15 \ge 7$, $1 \ge 7$ (X) swap end and start

7	6	7	5	1	2	9	15	10

start start end start end

$1 \le 7$, $2 \le 7$, $9 \le 7$ (X) stop start

$9 \ge 7$, $2 \ge 7$ (X)

stop > end swap pivot and end.

0	1	2	3	4	5	6	7	8
2	6	7	5	1	7	9	15	10

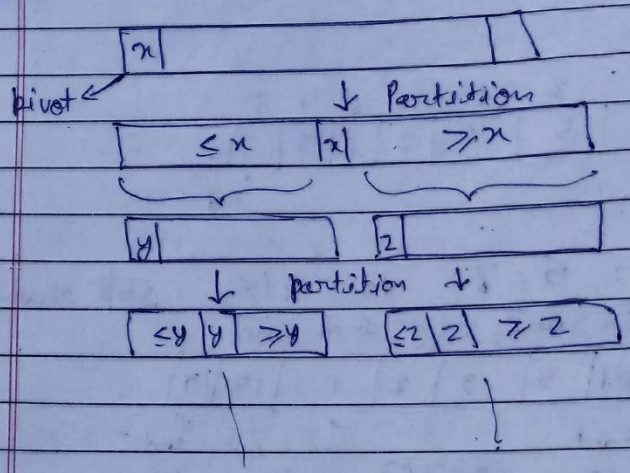
pivot

- **Divide**- Partition the array into two sub arrays around a pivot x such that elements in lower subarray $\le x \le$ elements in upper subarray.

$\le x$	x	$\ge x$

Page No.: 36
Date: / /

- Divide - (Partition).



- Conquer - Recursively sort the two subarrays.
- Combine - Trivial.

Key - Linear-time partitioning subroutine. $\Theta(n)$

→ Pseudo Code -

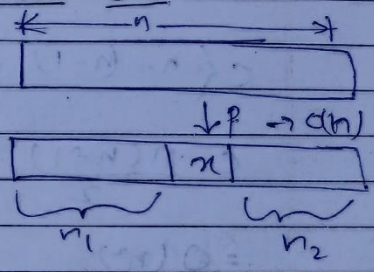
```
PARTITION(A, p, q)
1)  x ← A[p]
2)  i ← p
3)  for j ← p+1 to q
4)      do if A[j] ≤ x
5)          then i ← i+1
6)               exchange A[i] ↔ A[j]
7)  exchange A[p] ↔ A[i]
8)  return i
```

Page No.: 37
Date: / /

```
QUICK_SORT (A, p, r)
1) if p<r
2)     then q ← PARTITION(A, p, r)
3)          QUICKSORT(A, p, q-1)
4)          QUICK_SORT(A, q+1, r)
```

Note: initial call - Quick sort (A, 1, n).

→ Time Complexity of Quick Sort -

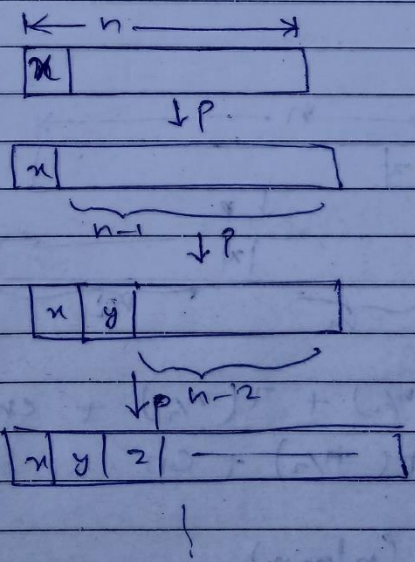


$T(n) = T(n_1) + T(n_2) + cn$

$n_1 + n_2 + 1 = n$.

$n_1 + n_2 \approx n$.

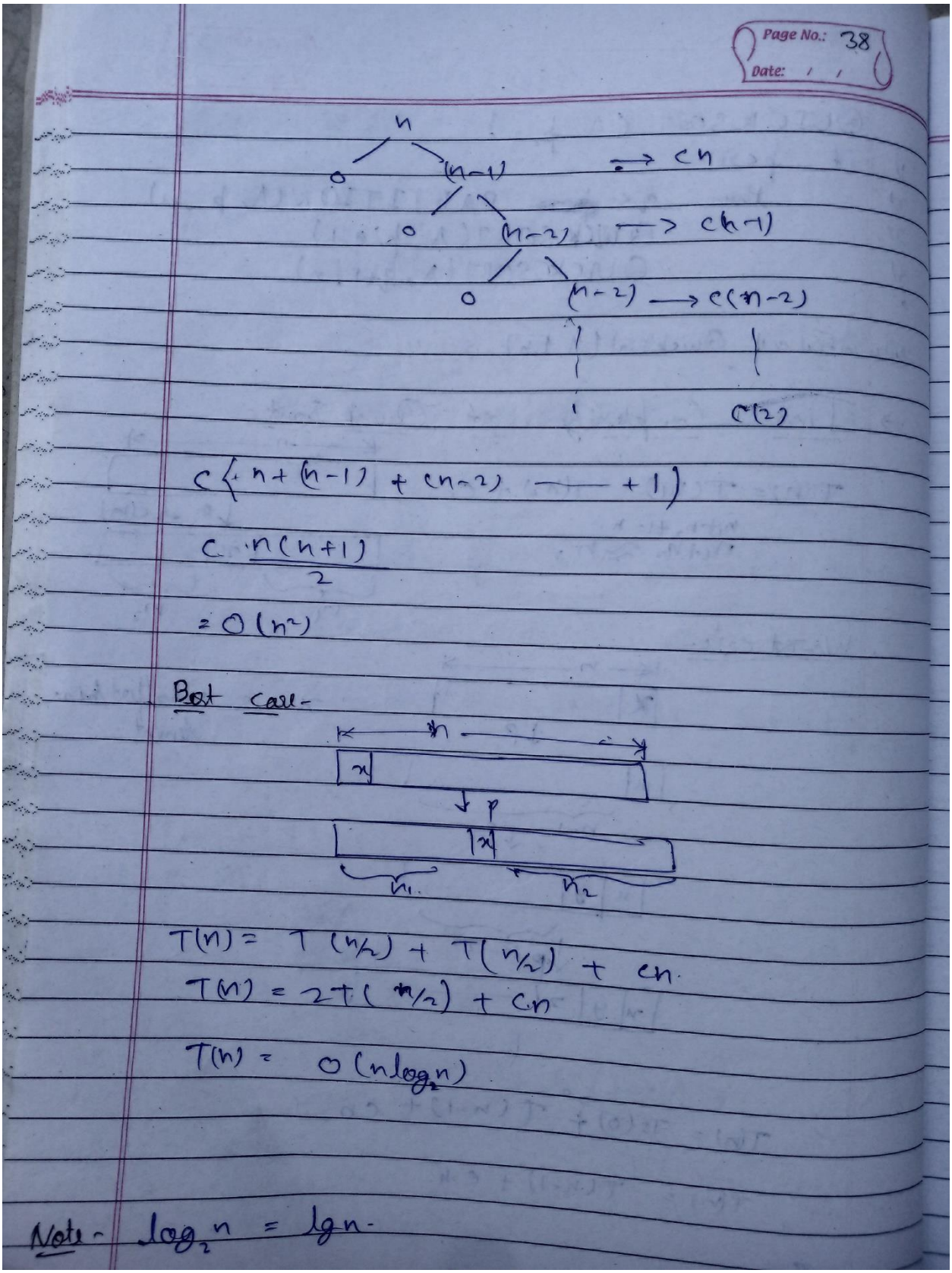
. Worst case -



x - is smallest/min. element.

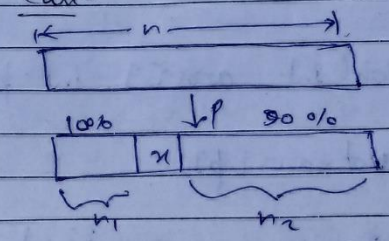
$T(n) = T(0) + T(n-1) + cn$

$T(n) = T(n-1) + c \cdot n$.


Page No.: 38
Date: / /
n
(n-1)
cn
(n-2)
c(n-1)
(n-2)
c(n-2)
c(2)
c{n+(n-1)+(n-2) ---- +1}
c n(n+1)/2
= O(n²)
Best case-
n
x
p
x
n1
n2
T(n) = T(n/2) + T(n/2) + cn
T(n) = 2T(n/2) + cn
T(n) = O(n log n)
Note- log2 n = lg n.

Page No.: 39
Date: / /

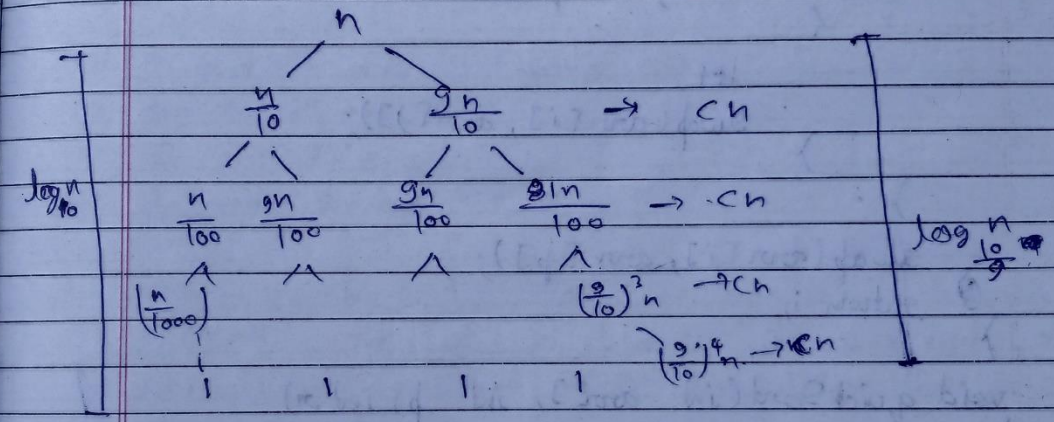
almost Best Case-



$n_1 \approx \frac{n}{10}$

$n_2 \approx \frac{9n}{10}$

$T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + cn$



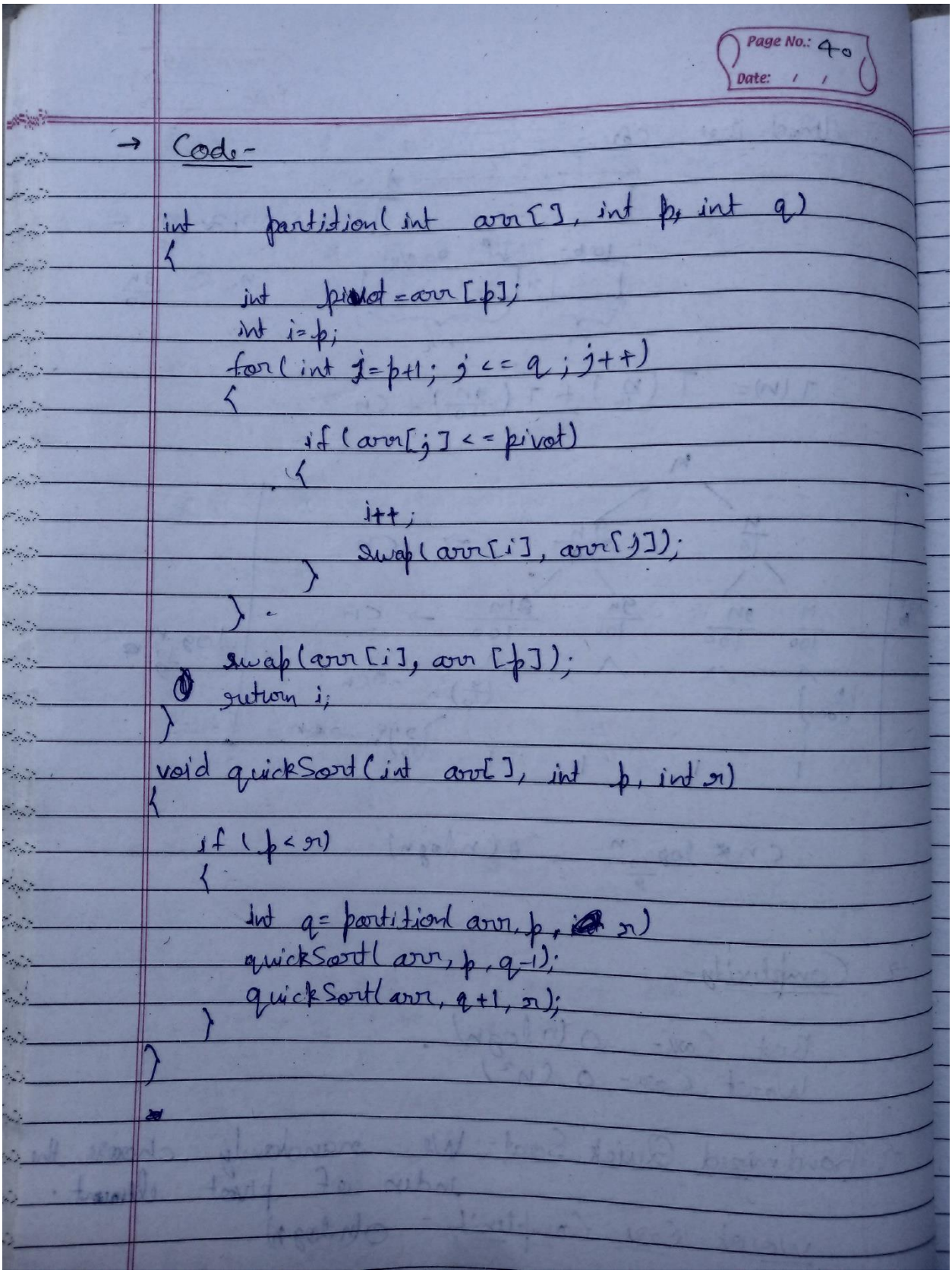
$cn * \log_{\frac{10}{9}} n = \theta(n \log n)$

→ Complexity-

Best Case- $O(n \log n)$
Worst Case- $O(n^2)$

→ Randmized Quick Sort- We randomly choose the index of pivot element.

Worst Case Complexity- $O(n \log n)$



Page No.: 40
Date: / /

→ Code-

```
int partition(int arr[], int p, int q)
{
    int pivot = arr[p];
    int i = p;
    for(int j = p+1; j <= q; j++)
    {
        if(arr[j] <= pivot)
        {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i], arr[p]);
    return i;
}
void quickSort(int arr[], int p, int r)
{
    if(p < r)
    {
        int q = partition(arr, p, r)
        quickSort(arr, p, q-1);
        quickSort(arr, q+1, r);
    }
}
```

Page No.: 41
Date: 26/10/20

Q- Assume that the algorithms considered here sort the input sequences in ascending order. If the input is already in ascending order, which of the following are True?

(i) Quick runs in $\Theta(n^2)$ time
(ii) Bubble sort runs in $\Theta(n^2)$ time
(iii) Merge sort runs in $\Theta(n)$ time
(iv) Insertion sort runs in $\Theta(n)$ time

(A) I and II only
(B) I and III only
(C) II and IV only
(D) I and IV only (✓)

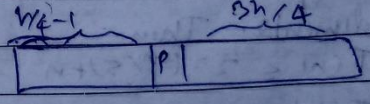
Q- In quick-sort, for sorting n elements, the $(n/4)$th smallest element is selected as pivot using an $O(n)$ time algorithm.

(i) $\Theta(n)$
(ii) $\Theta(n \log n)$ (✓)
(iii) $\Theta(n^2)$
(iv) $\Theta(n^2 \log n)$

Selecting $(n/4)^{th}$ element - $O(n)$ ↓ ↗ Partition

$$T(n) = O(n) + O(n) + T\left(\frac{n}{4} - 1\right) + T\left(\frac{3n}{4}\right)$$

$T(n) = \Theta(n \log n)$.



Page No.: 42
Date: / /

Q- Randomized quicksort is an extension of quicksort where the pivot is chosen randomly. What is the worst case complexity of sorting n numbers using Randomized quicksort?

(i) $O(n)$
(ii) $O(n*\log n)$
(iii) $O(n^2)$ ✓
(iv) $O(n!)$

Case 1- all the elements are same-

3, 3, 3, 3, 3, 3. Then

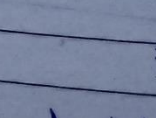
$T(n) = T(n-1) + T(0) + \theta(n)$

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

Case 2- V. V. small chance always picking the smallest element as pivot. Then $T(n) = O(n^2)$

Q- Consider the Quicksort algorithm. Suppose there is a procedure for finding a pivot element which splits the list into two sub-lists each of which contains at least one-fifth of the elements. Let $T(n)$ be the number of comparisons required to sort n elements. Then.

(i) $T(n) \leq 2T(n/5)+n$
(ii) $T(n) \leq T(n/5) + T(4n/5) + n$ ✓
(iii) $T(n) \leq 2T(4n/5) + n$
(iv) $T(n) \leq 2T(n/2) + n$



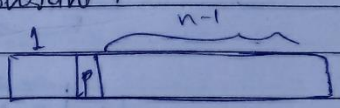
partition

Page No.: 43
Date: / /

Q- Which one of the following is the recurrence equation for the worst case time complexity of the quick sort algorithm for sorting $n(\geq 2)$ numbers? In the recurrence equations given in the options below, c is a constant.

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

(ii) $T(n) = T(n-1) + T(1) + cn$ (✓)



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

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

Q- You have an array of n elements. Suppose you implement quicksort by always choosing the central element of the array as the pivot. Then the tightest upper bound for the worst case performance is-

(i) $O(n^2)$ (✓)

(ii) $O(n \log n)$

(iii) $\Theta(n \log n)$

(iv) $O(n^3)$

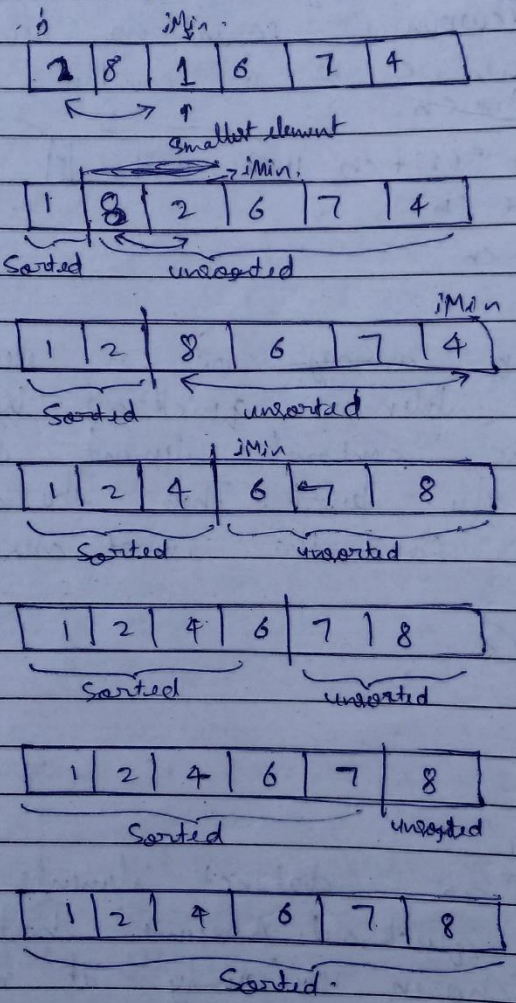
Q- An array of 25 distinct elements is to be sorted using quicksort. Assume that pivot element is chosen uniformly at random. The probability that the pivot element gets placed in the worst possible location in the first round of partitioning (rounded off to 2 decimal places) is- Worst position is max. and min index and there is unique max. and min. then probability $= \frac{2}{25} = 0.08$

Page No.: 44
Date: / /

Selection Sort -

→ Intuition - Select the smallest element in the left over array.

Eg -



Page No.: 45
Date: / /

→ Code-

```
void selectionSort(int arr[], int n)
{
    for(int j=0; j<n-1; j++)
    {
        int iMin = j;
        for(int i=j+1; i<n; i++)
            if (arr[i] < arr[iMin])
                iMin = i;
        if (iMin != j)
            swap(arr[j], arr[iMin]);
    }
}
```

Time & Space Complexity-

j = 0
(n-1) Comparisions and 1 swap
j = 1
(n-2) Comparisions and 1 swap
j = 2
(n-3) Comparisions and 1 swap

= (n-1) + (n-2) + ——— + 1 (comparisions)
= O(n²) comparisions

1 + 1 + 1 + ——— + 1 n swaps
= O(n) swap

Time Complexity - O(n²)
Space Complexity - O(1) (Inplace Sorting Algorithm)

Page No.: 46
Date: / /

In worst Case -

Insertion sort - $O(n^2)$ Swaps and $O(n^2)$ Comp.

Selection Sort - $O(n^2)$ Swaps and $O(n^2)$ Comp.

Note - If time to read < time to write then we use selection sort because swap is a write operation and number of swap is $O(n)$ in selection sort and no. of ~~Swap~~ Comp $O(n^2)$.

Q- Which one of the following is the tightest upper bound that represents the number of swaps required to sort n numbers using selection sort?

(i) $O(\log n)$
(ii) $O(n)$ (✓)
(iii) $O(n \log n)$
(iv) $O(n^2)$

Q- Which of the following sorting algorithms has the lowest worst-case complexity?

(i)	Merge Sort (✓)	$O(n \log n)$
(ii)	Bubble Sort	$O(n^2)$
(iii)	Quick Sort	$O(n^2)$
(iv)	Selection Sort	$O(n^2)$

Page No.: 47
Date: / /

Q- The worst case ~~xxxx~~ runing times of insertion sort, Merge sort and Quicksort respectively, are-

(i) $\theta(n \log n)$, $\theta(n \log n)$ and $\theta(n^2)$
(ii) $\theta(n^2)$, $\theta(n^2)$ and $\theta(n \log n)$
(iii) $\theta(n^2)$, $\theta(n \log n)$ and $\theta(n \log n)$
(iv) $\theta(n^2)$, $\theta(n \log n)$ and $\theta(n^2)$ (✓)

Q- Which of the following sorting algorithm has a running time that is least dependent on the initial ordering of the inputs?

(i) Quick Sort → $O(n \log n)$ (best), $O(n^2)$ (worst)
(ii) Insertion Sort → $O(n)$, $O(n^2)$
(iii) Merge Sort (✓) → $O(n \log n)$
(iv) Selection Sort → $\Omega(n^2)$, $\theta(n^2)$

Page No.: 48
Date: 27/10/20

Linear time Sorting -

Q - Can there be any comparison sorting algo worst case time $O(n)$.

Let $A = [a_1, a_2, a_3, \dots, a_n]$ $a_i \neq a_j \ \forall \ i, j$

$n = 3$

$\langle a_1, a_2, a_3 \rangle$ how many to arrange 3 numbers

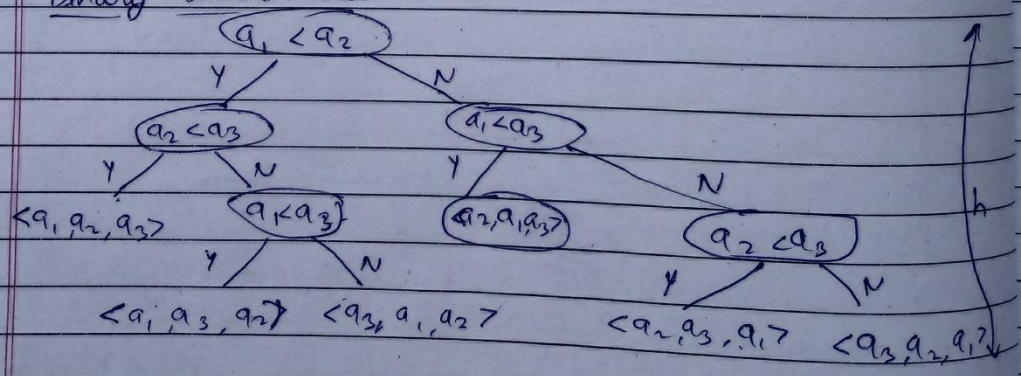
1, 2, 3
1, 3, 2
2, 1, 3
2, 3, 1
3, 1, 2
3, 2, 1

$6 = 3! = 3 \times 2 \times 1$.

for n numbers arrange $n!$ ways.

$n \longrightarrow n!$ — (1).

Binary Decision Tree -



No of leaf-nodes = no. of arrangements.

no. of leaf node = $n!$

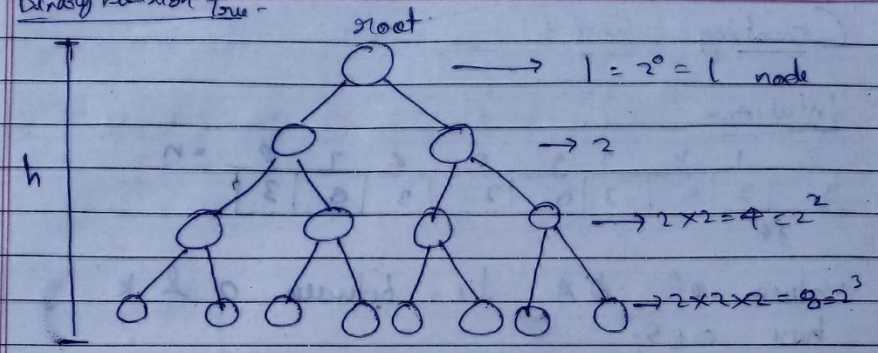
for n-elements $\longrightarrow$ $n!$ arrangements.

h = no. comparisions we have to make to sort

B.D.T. is a mathematical abstract model for comparison based sorting

Page No.: 49
Date: / /

Binary Decision Tree -



If a binary decision tree has height h
max no. of leaf nodes = 2^h

If h is the height of the binary D.T. to sort n elements:

$$2^h \geq n!$$

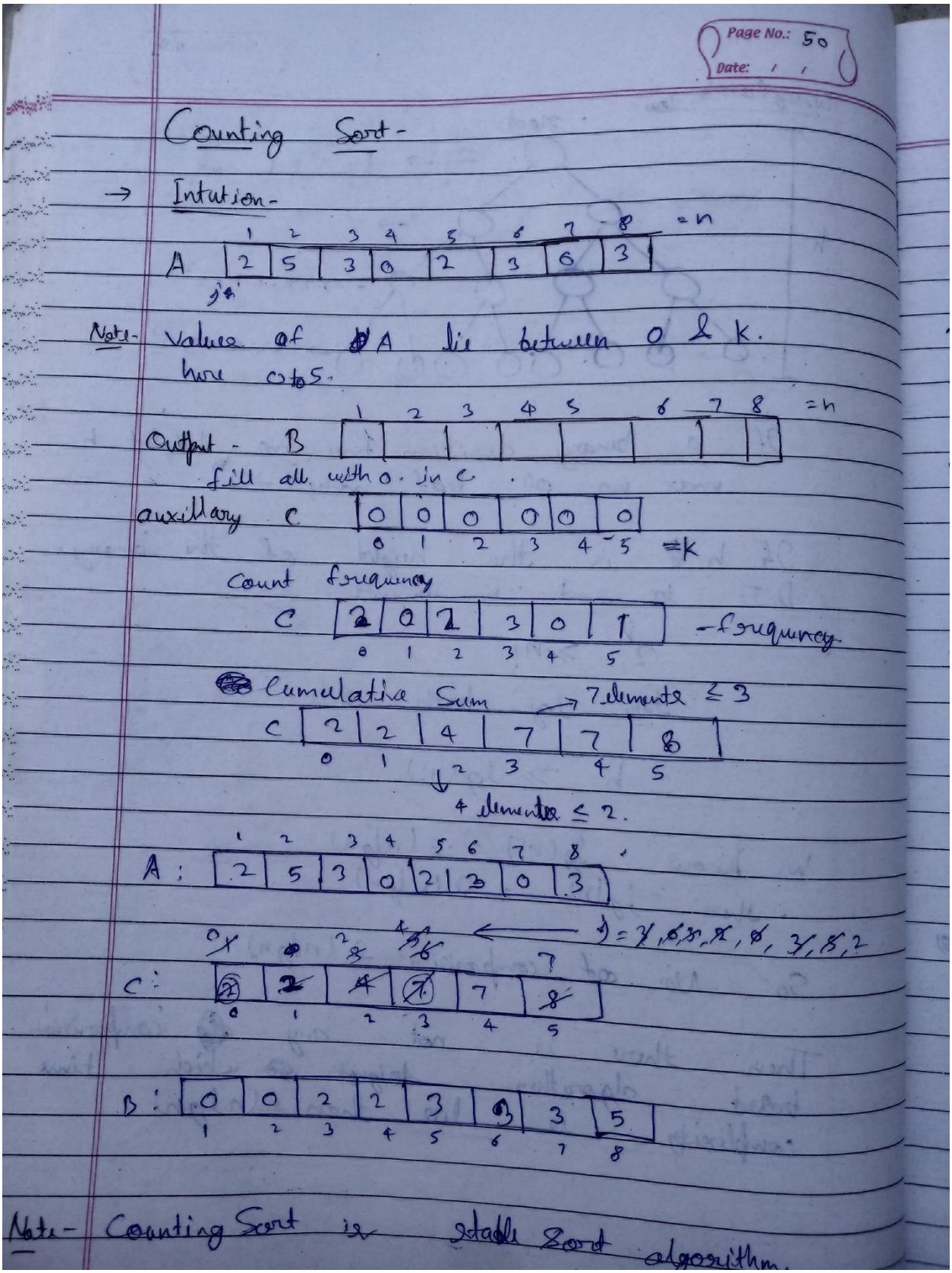
take log both side -

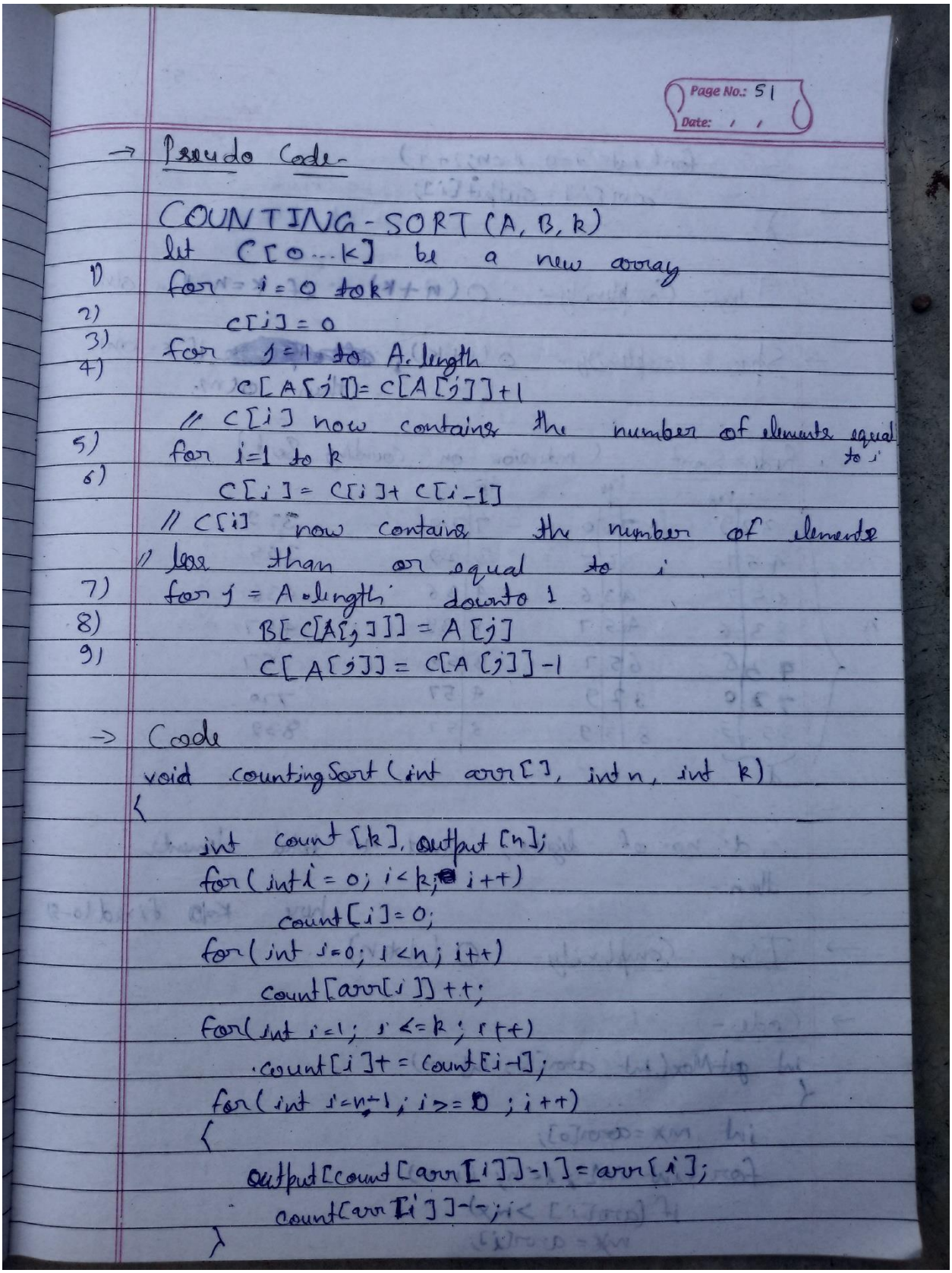
$$h \geq \lg(n!)$$

We know $\lg(n!) = \theta(n \lg n)$
then $\lg(n!) = \Omega(n \lg n)$

So. No. of comparision = $\Omega(n \lg n)$

Then there is not any comparision based algorithm designed which time complexity is less than $\Omega(n \lg n)$


Page No.: 50
Date: / /
Counting Sort -
→ Intution -
A 2 5 3 0 2 3 0 3 = n
Note - Values of A lie between 0 & k.
here 0 to 5.
Output - B = n
fill all with 0 in C.
auxillary C 0 0 0 0 0 0 = k
Count frequency
C 2 0 2 3 0 1 - frequency
Cumulative Sum → 7 elements ≤ 3
C 2 2 4 7 7 8
4 elements ≤ 2.
A: 2 5 3 0 2 3 0 3
C: 2 2 4 7 7 8
B: 0 0 2 2 3 3 3 5
Note - Counting Sort is stable Sort algorithm.



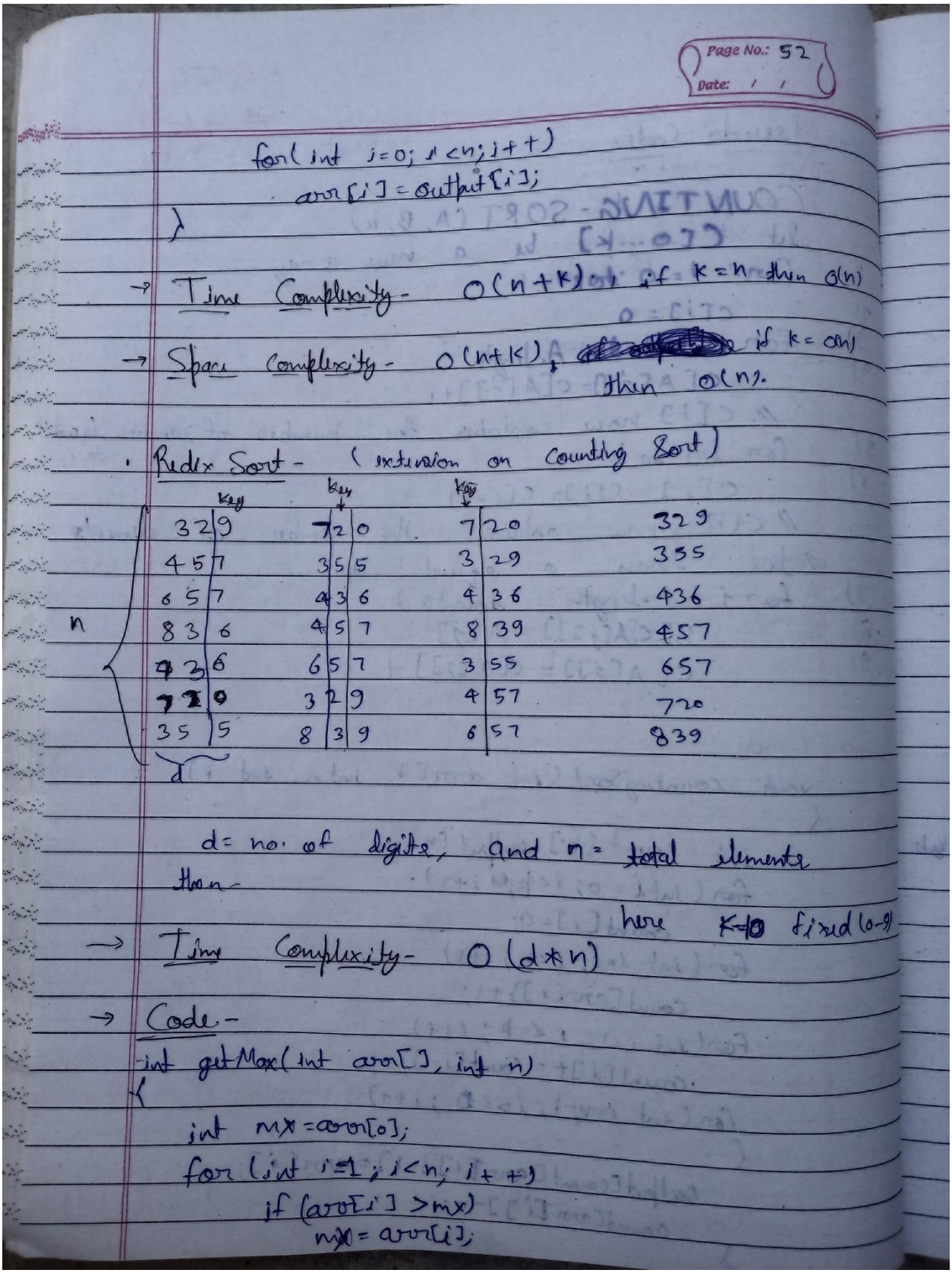
Page No.: 51
Date: / /

→ Pseudo Code-

```
COUNTING-SORT (A, B, k)
let C[0...k] be a new array
1) for i = 0 to k
2)     C[i] = 0
3) for j = 1 to A.length
4)     C[A[j]] = C[A[j]] + 1
   // C[i] now contains the number of elements equal to i
5) for i = 1 to k
6)     C[i] = C[i] + C[i-1]
   // C[i] now contains the number of elements
   // less than or equal to i
7) for j = A.length downto 1
8)     B[C[A[j]]] = A[j]
9)     C[A[j]] = C[A[j]] - 1
```

→ Code

```
void countingSort(int arr[], int n, int k)
{
    int count[k], output[n];
    for(int i = 0; i < k; i++)
        count[i] = 0;
    for(int j = 0; j < n; j++)
        count[arr[j]]++;
    for(int i = 1; i <= k; i++)
        count[i] += count[i-1];
    for(int i = n-1; i >= 0; i++)
    {
        output[count[arr[i]]-1] = arr[i];
        count[arr[i]]--;
    }
```



Page No.: 52
Date: / /

```
for(int i=0; i<n; i++)
    arr[i] = output[i];
}
```

→ Time Complexity - O(n+k) ; if k=n then O(n)

→ Space Complexity - O(n+k) ; if k=n/ then O(n).

• Radix Sort - (extension on Counting Sort)

key		key		key	
329		720		720	
457		355		329	
657		436		436	
839		457		839	
436		657		355	
720		329		457	
355		839		657	

n (rows) × d (columns)

d = no. of digits, and n = total elements

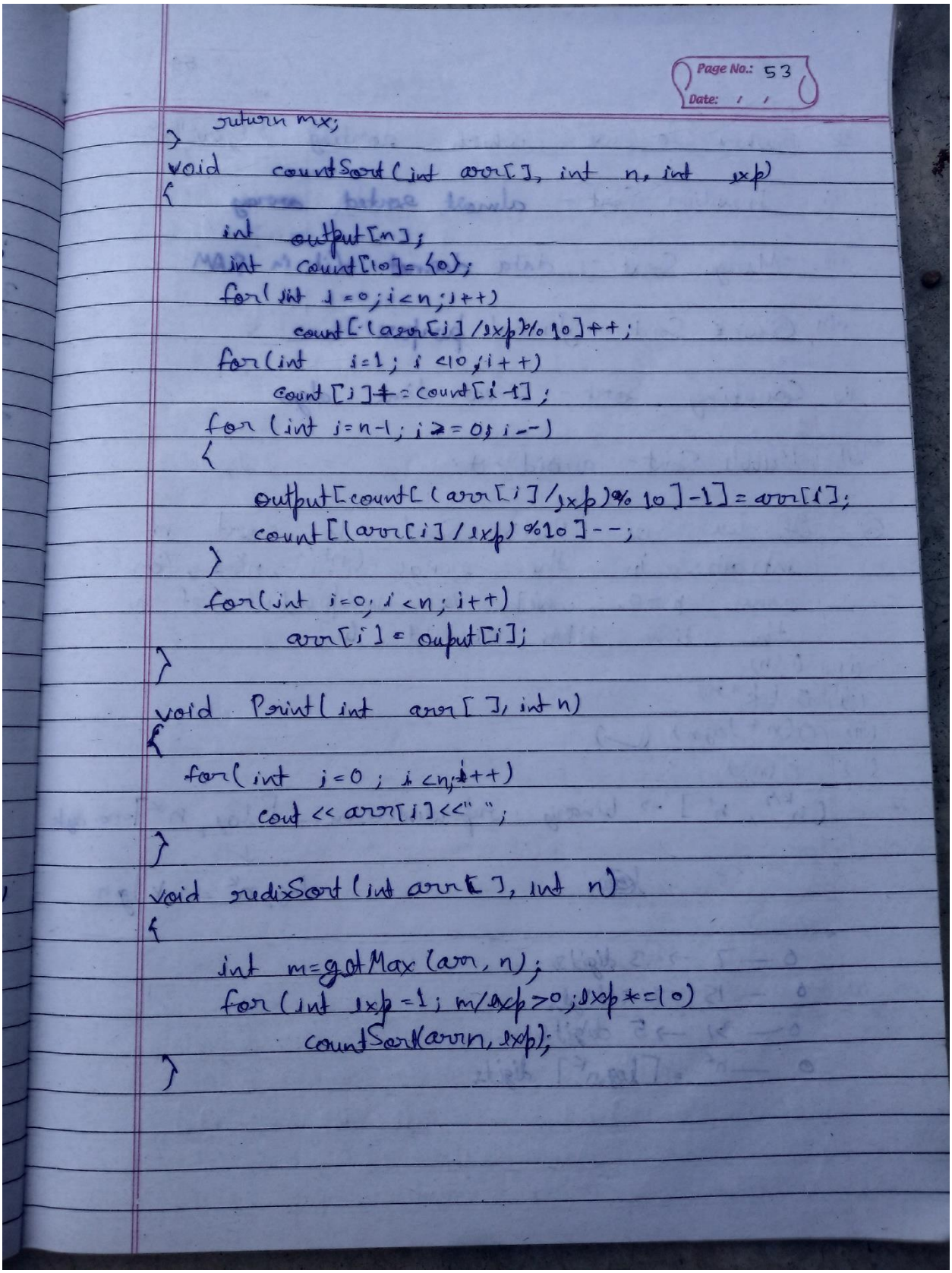
then -

here K=10 fixed (0-9)

→ Time Complexity - O(d*n)

→ Code -

```
int getMax(int arr[], int n)
{
    int mx = arr[0];
    for (int i=1; i<n; i++)
        if (arr[i] > mx)
            mx = arr[i];
```



Page No.: 53
Date: / /

```
    return mx;
}
void countSort(int arr[], int n, int exp)
{
    int output[n];
    int count[10] = {0};
    for(int i=0; i<n; i++)
        count[(arr[i]/exp)%10]++;
    for(int i=1; i<10; i++)
        count[i] += count[i-1];
    for (int i=n-1; i>=0; i--)
    {
        output[count[(arr[i]/exp)%10]-1] = arr[i];
        count[(arr[i]/exp)%10]--;
    }
    for(int i=0; i<n; i++)
        arr[i] = output[i];
}
void Print(int arr[], int n)
{
    for(int i=0; i<n; i++)
        cout << arr[i] << " ";
}
void radixSort(int arr[], int n)
{
    int m = getMax(arr, n);
    for (int exp=1; m/exp>0; exp*=10)
        countSort(arr, exp);
}
```

Page No.: 54
Date: / /

Q- Where to use which sorting Algorithm?

(i) Insertion Sort - almost sorted array

(ii) Merge Sort - data doesn't fit in RAM

(iii) Quick Sort - General purpose

(iv) Counting Sort - Linear time algo

(v) Bubble Sort - avoid it.

Q- If we use Radix Sort to sort n integers in the range $(n^{k/2}, n^k]$, for some $k > 0$, which is independent of n, the time taken would be?

(i) $O(n)$

(ii) $O(k*n)$

(iii) $O(n*\log n)$ (✓)

(iv) $O(n^2)$

$(n^{k/2}, n^k]$ -> binary representation $= \lceil \log_2 n^k \rceil$ = no. digits

$\log_2 n^k = k \log n$

0 — 7 → 3 digits

0 — 15 → 4 digits

0 — 31 → 5 digits

0 — $n^c = \lceil \log_2 n^k \rceil$ digits

Data Structure

Page No.: 55
Date: 28/10/20

Data Structure - A data structure is a collection of data values, the relationships among them and the functions or operations that can be applied to the data.
Ex- Arrays, LinkedLists, Trees, Graphs etc.

Eg- Array A - | 6 | 8 | 7 | 9 | 10 |
1 2 3 4 5

(i) Collection of items of same data type
(ii) Ordering of data
(iii) Operations - set, get, traverse, index search.

- **Array** - Collection of items of similar type data.

1- **1-D Array** - Length = UB - LB + 1

Eg- int a[10];. (contigious memory location.)

a = ← 1000 1004 1008 1012 — — — — — —

Value ← | a_0 | a_1 | a_2 | a_3 | a_4 | a_5 | a_6 | a_7 | a_8 | a_9 |
← 4 bytes

i = 0 1 2 3 4 5 6 7 8 9 ← index

K → 40 Bytes → 4×10 = 40 Bytes

a - is pointer to a_0 [1000]
a.

address of a_i = 1000 + (i * 4).

Loc [A[K]] = Base(A) + W(K - LB) W = No. byte for each location value.

1000 - 1003 → a_0 (4B)
1004 - 1007 → a_1

→ **Pointer arithmatic** -
int *p = a; So p = 1000 = a[0] = a.
then p+1 = 1004 = a[1], p+2 = 1008 = a[2], and soon

Page No.: 56
Date: / /

2- 2D - Array

Eg int a[5][4]
(5 = rows, 4 = columns)

	Columns			
	a_{00}	a_{01}	a_{02}	a_{03} → a[0][3]
	a_{10}	a_{11}	a_{12}	a_{13} → a[1][3]
grow	a_{20}	a_{21}	a_{22}	a_{23} → a[2][3]
	a_{30}	a_{31}	a_{32}	a_{33} → a[3][3]
	a_{40}	a_{41}	a_{42}	a_{43} → a[3][4]

→ Memory Organized -

(i) Row Major Order (RMO)
(ii) Column Major Order (CMO) (FORTRAN Language)

Eg- n × m
(row) × (column)

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ ——— $a_{0,m-1}$
$a_{1,0}$ $a_{1,1}$ ——— $a_{1,m-1}$
$a_{2,0}$ $a_{2,1}$ ——— $a_{2,m-1}$
$a_{n-1,0}$ $a_{n-1,1}$ ——— $a_{n-1,m-1}$

→ Row Major Order

1D
1000 1004 1008

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	—	$a_{0,m-1}$	$a_{1,0}$	$a_{1,2}$	—	$a_{1,m-1}$	$a_{2,0}$	—

1 row 2nd row.

memory loc $(a_{i,j})$ = 1000 + (m * i) * 4 + j * 4

LOC (A[i, k]) = Base(A) + w[N.(i-1) + (k-1)]

where w = No. Bytes required for each value.
N = No. rows.

Page No.: 57
Date: / /

→ Column Major Order (CMO)-

1004

$a_{0,0}$	$a_{1,0}$	$a_{2,0}$	...	$a_{n-1,0}$	$a_{0,1}$	$a_{1,1}$	$a_{2,1}$	...	$a_{n-1,1}$	$a_{0,2}$	$a_{1,2}$	...

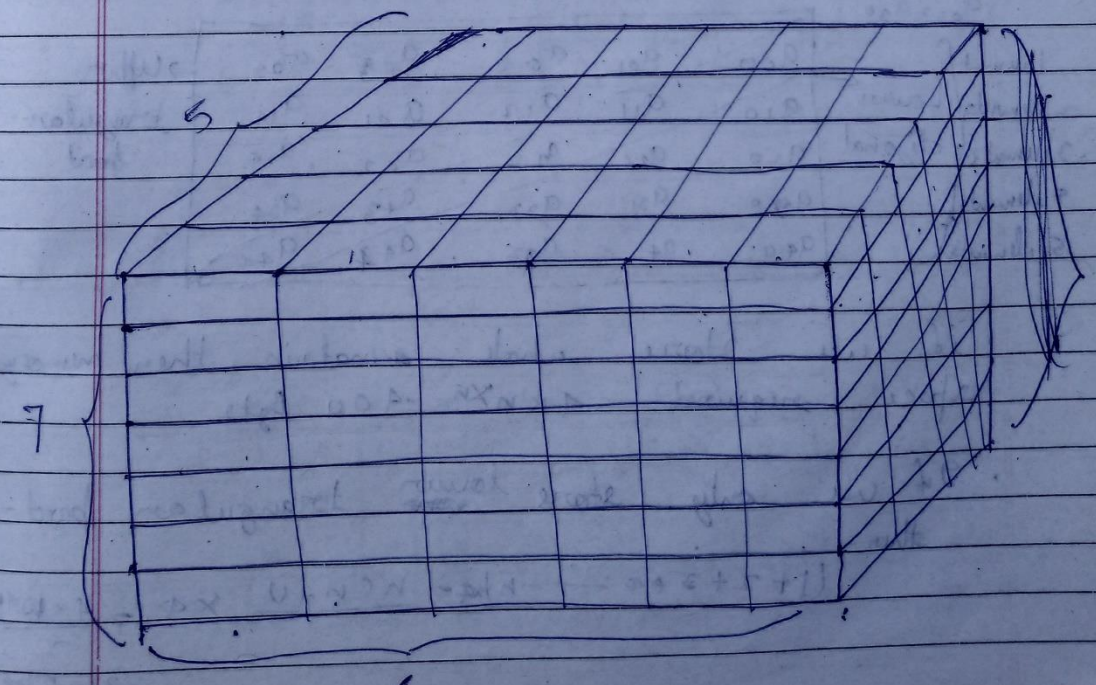
1st Column 2nd Column.

mem_loc $(a_{i,j})$ = 1000 + (j * n) * 4 + (i * 4)

Note- When we perform any operation on rows then row major order is better and if we perform any operation on columns then column major order is better.

Multi-D Array-

Eg- int A[7][6][5]



Page No.: 58
Date: / /

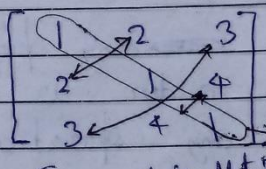
Spacial Matrices - (2D Array).-

1- Square Matrix - No. of rows = No. columns ($n \times n$)

2- digonal → $a_{i,i}$

3- Symmetric Matrix - $a_{ij} = a_{ji}$ $\forall i,j$

Eg-



Symmetric Matrix 3×3

Note- In case of Symmetric Matrix we only store lower digonal elements or upper digonal elements because $a_{ij} = a_{ji}$ $\forall i,j$

$a_{03} = a_{30}$

1 element		a_{00}	a_{01}	a_{02}	a_{03}	a_{04} → Upper triangular part
2 elements	Lower digonal	a_{10}	a_{11}	a_{12}	a_{41}	a_{14}
3 elements		a_{20}	a_{21}	a_{22}	a_{42}	a_{24}
4 elements		a_{30}	a_{31}	a_{32}	a_{43}	a_{34}
5 elements		a_{40}	a_{41}	a_{42}	a_{44}	a_{44}

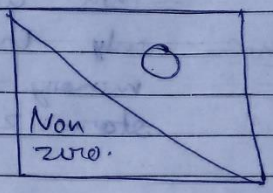
- if we store whole matrix then memory space required - $4 \times n^{\times n} = 400$ Byte

- If we only store lower triangular part - then -

$1+2+3+4 \cdots n/4 = \frac{n(n+1)}{2} \times 4 = \frac{4 \times 10 \times 11}{2}$

Page No.: 59
Date: / /

4- Lower triangular Matrix-

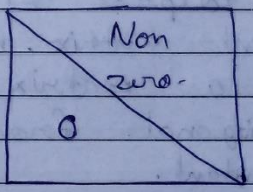


Eg-

2	0	0
7	4	0
0	8	5

store only lower triangular part.
$= \frac{n(n+1)}{2}$

5- Upper triangular Matrix-

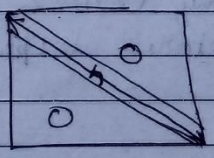


Eg-

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

store only upper triangular part
$= \frac{n(n+1)}{2}$

6- Digonal Matrix-



Eg-

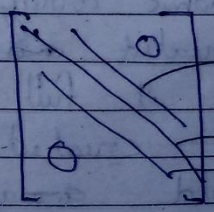
1	0	0
0	2	0
0	0	3

store only digonal part- $= n$.

7- Tridiagonal Matrix- has non-zero elements only on the main diagonal, the first diagonal, the first diagonal below this, and the first diagonal above the main diagonal.

Eg-

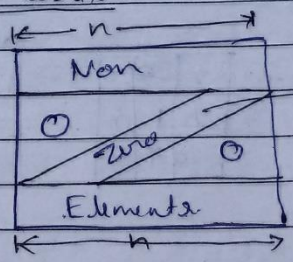
1	4	0	0
3	2	7	0
0	6	5	10
0	0	9	8



required memory space to store $= 2(n-1)+n = 3n-2$

Page No.: 60
Date: / /

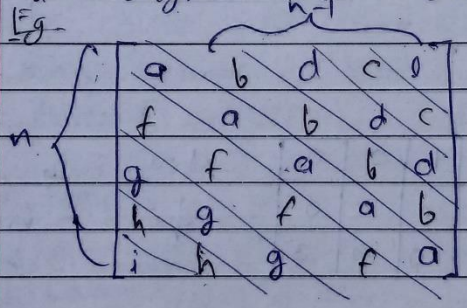
8- Z-Matrix-



Then we required only $(n-2)+2n = (3n-2)$ memory space to store Z-matrix.

9- Toeplitz Matrix- A toeplitz or diagonal-constant matrix, named after Otto Toeplitz, is a matrix in which each descending diagonal from left to right is constant.

Eg-



So we required only $(2n-1)$ space to store toeplitz Matrix.

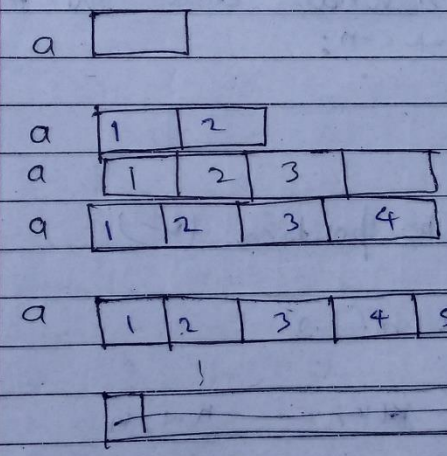
Dynamic Array- The size of array increase or decrease at the Run time.

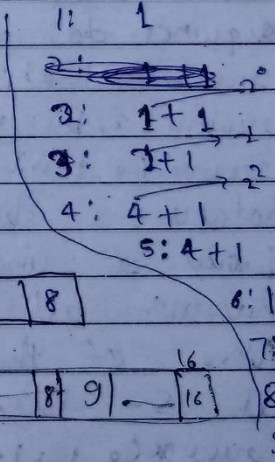
- In C++ has vector and Java has Arraylist for dynamic array and in C we implement that.
- In Dynamic array first a fixed size array is created suppose size is n and whenever it full a new array of size 2n is created and copy the element of old array to new array.
- In case of static array complexity of

Page No.: 61
Date: / /

insertion is O(1) but in case of dynamic array time complexity of insertion is O(1) in best case if initial size of array is not full and O(k) in worst case if initial array is full. (due to copying). where k is total no. of element in initial array.

→ Amortized analysis - It comput cost of 'insert' (any) operation which is averaged over many such operations





n=17

$= (17*1) + \{2^0 + 2^1 + 2^2 + 2^3 + 2^4\}$

$= (n+1) + \{2^0 + 2^1 + \dots + 2^k\}$ such that $k = \lfloor \log_2 n \rfloor$

$= n + O(n) = O(n)$.

O(n) → n insertions.
O(1) → 1 insertion.

$2^0 + 2^1 + 2^2 + \dots + 2^k$
$= 2^{k+1} - 1 \Rightarrow O(n)$.
$= 2 2^k + 1 \approx 2^k = O(n)$

Page No.: 62
Date: / /

Q- An n×n array v is defined as follows- $V[i,j] = i-j$ for all i, j such that $i<n$, $1 \le j \le n$. The sum of the elements of the array v is-

Eg Let n=3

0	-1	-2
1	0	-1
2	1	0

$= 0$.

(i) 0 (✓)
(ii) n-1
(iii) n^2-3n+3
(iv) $n^2(n+1)/2$

Q- Suppose you are given an array s[1...n] and a procedure reverse(s, i, j) which reverses the order of elements in a between positions i and j (both inclusive). What does the following sequence do, where $1 <= k <= n$;

reverse(s, 1, k);
reverse(s, k+1, n);
reverse(s, 1, n)

(i) Rotates s left by k positions (✓)
(ii) Leaves s unchanged
(iii) Reverses all elements of s
(iv) None of the above.

Eg Let S- 1, 2, 3, 4, ... k, k+1, ... n-1, n

reverse(s, 1, k)-

k, k-1, ... 3, 2, 1, k+1, k+2, ... n-1, n

reverse(s, k+1, n)-

k, k-1, ... 2, 1, n, n-1, ... k+2, k+1

reverse(s, 1, n)

k+1, k+2, ... n-1, n, 1, 2, 3 ... k-1, k.

Page No.: 63
Date: 29/10/20

Q- A single array A[1 --- MAXSIZE] is used to implement two stacks. The two stacks grow from opposite ends of the array. Variables top1 and top2 (top1 < top2) point to the location of the topmost element in each of the stacks. If the space is to be used efficiently, the condition for "stack full" is

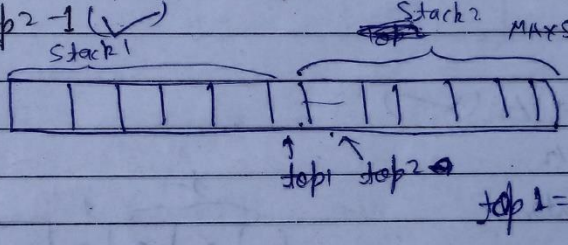
(i) (top1 = MAXSIZE/2) and (top2 = MAXSIZE/2 + 1)

(ii) top1 + top2 = MAXSIZE

(iii) (top1 = MAXSIZE/2) or (top2 = MAXSIZE)

(iv) top1 = top2 - 1 ✓

Eg -



top1 = top2 - 1

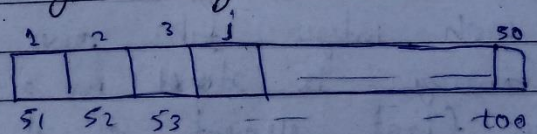
Q- A program P reads in 500 integers in the range [0 - 100] representing the scores of 500 students. It then prints the frequency of each score above 50. What would be the best way for P to store the frequencies?

(i) An array of 50 numbers (—)

(ii) An array of 100 numbers

(iii) An array of 500 numbers

(iv) A dynamically allocated array of 550 numbers



freq[50+i]

Page No.: 64
Date: / /

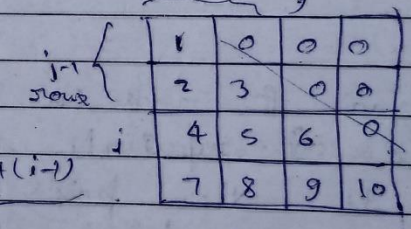
Q- In a compact single dimensional array representation for lower traingular matrices (i.e. all the elements above the diagonal are zero). of size n×n, non-zero elements, (i.e. elements of lower triangle) of each row are stored one after another, starting from the first row, the index of the (i,j)th element of the lower triangular matrix in this new representation is-

(i) $i+j$
(ii) $i+j-1$
(iii) $(j-1)+i(i+1)/2$ (✓)
(iv) $i+j(j-1)/2$

row elements
1 : 1
2 : 2
3 : 3
⋮
i-1 : i-1

$1+2+3+\cdots+(i-1)$

$\frac{(i-1)i}{2}$



$\frac{(i-1)i}{2} + (j-1)$

Q- Let A be a two dimensional array declared as follows-
A: array[1....10][1....15] of integer
Assuming that each integer take one memory location, the array is stored in row-major order and the first element of the array is stored at location 100, what is the address of the element A[i][j]?

(i) $15i+j+84$ (✓)

Page No.: 65
Date: / /

(ii) $15j + i + 84$
(iii) $10i + j + 89$
(iv) $10j + i + 89$

		1	2	3	4	—	15
$(100 + (i-1)*15)$ ← 1st element ith row.	1	100	101	102	103	—	114
	2	115	116				129
$100(i-1)15 + (j-1)$	3	130					
jth element of ith row.	4	145	146	147	148		
	10						

$= 100 + 15i + j - 15 - 1$
$= 15i + j + 84$

Q- Two matrices M1 and M2 are to be stored in arrays A and B respectively. Each array can be stored either in row-major or column-major order in contiguous memory locations. The time complexity of an algorithm to compute M1 × M2 will be-

(i) best if A is in row-major, and B is in CMO
(ii) best if both are in row-major order
(iii) best if both are in column-major order
(iv) independent of the storage scheme. (✓)

if efficiency concern then it is true.

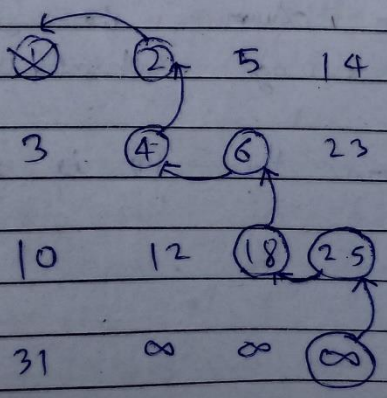
Page No.: 66
Date: / /

Q- A Young tableau is a 2D array of increasing from left to right and from top to bottom. Any unfilled entries are marked with ∞, and hence there cannot be any entry to the right of, or below a ∞. The following Young tableau consists of unique entries.

1	2	5	14
3	4	6	23
10	12	18	25
31	∞	∞	∞

When an element is removed from a Young tableau, other elements should be moved into its place so that the resulting table is still a Young tableau (unfilled entries may be filled in with a ∞). The minimum number of entries (other than 1) to be shifted, to remove 1 from the given Young tableau is ____

(i) 2
(ii) 5 (✓)
(iii) 6
(iv) 18



2, 4, 6, 18, 25

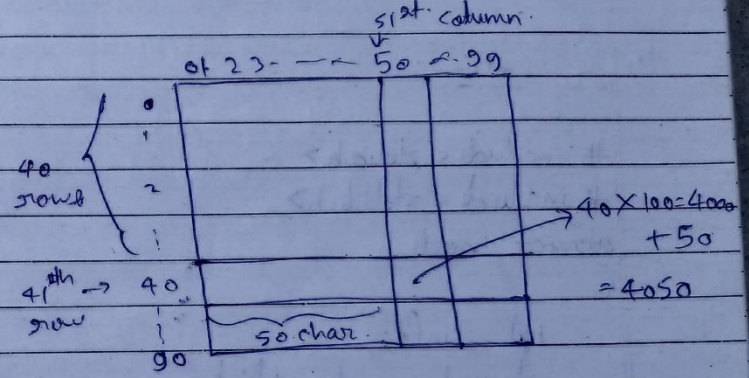
Page No.: 67
Date: / /

Q- Consider the following declaration of a two-dimensional array in C-
char a[100][100];
Assuming that the main memory is byte-addressable that the array is stored starting from memory address 0, the address of a[40][50] is -

(i) 4040
(ii) 4050 (✓)
(iii) 5040
(iv) 5050

1 byte

a[40][50]

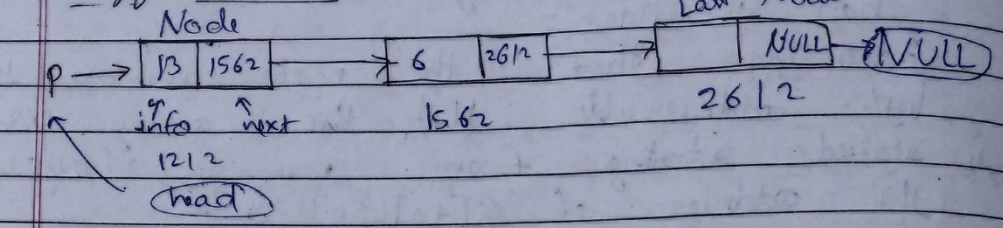


Page No.: 68
Date: 30/10/20

Linked List

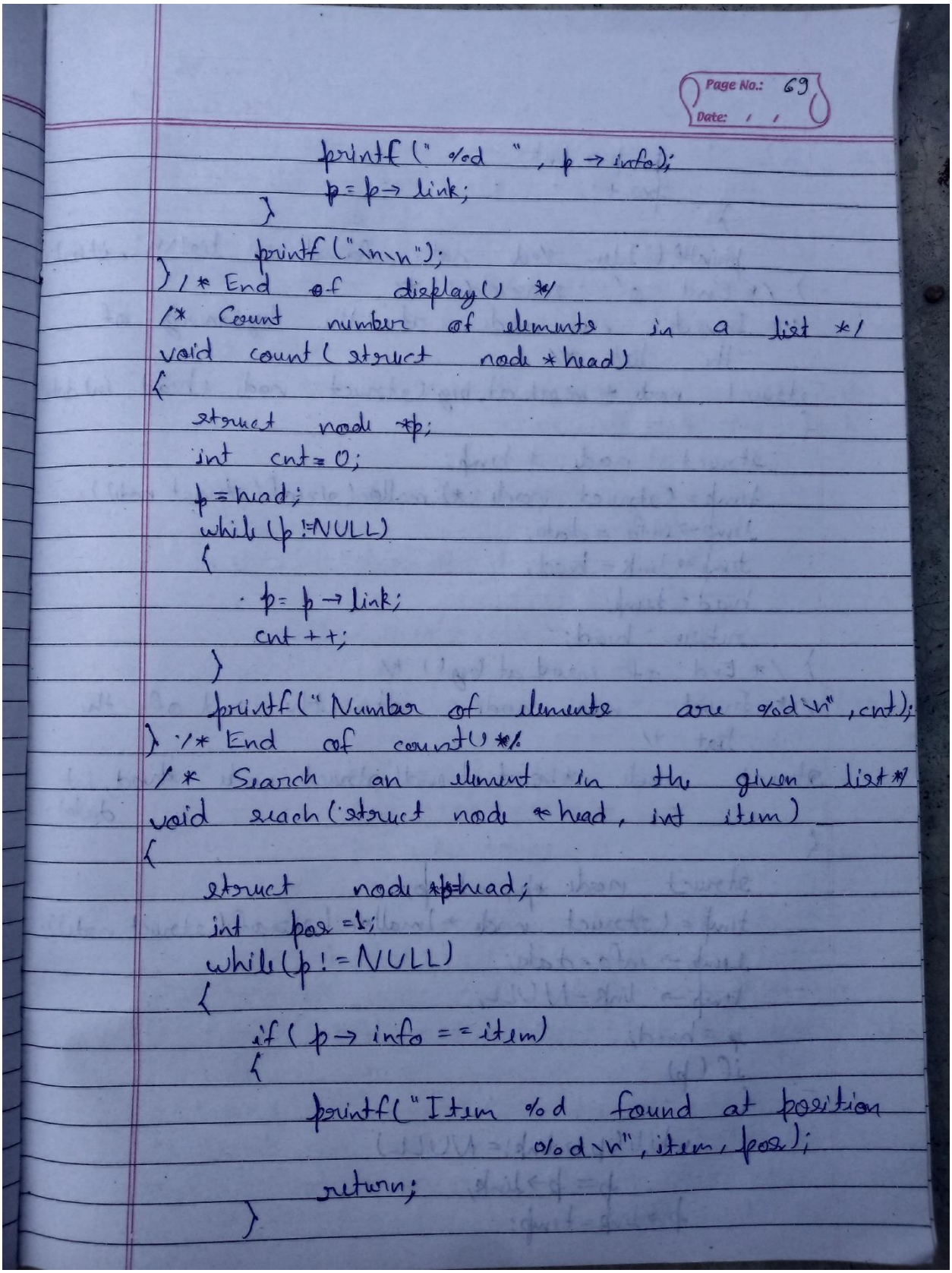
Linked List -

Singly Linked List

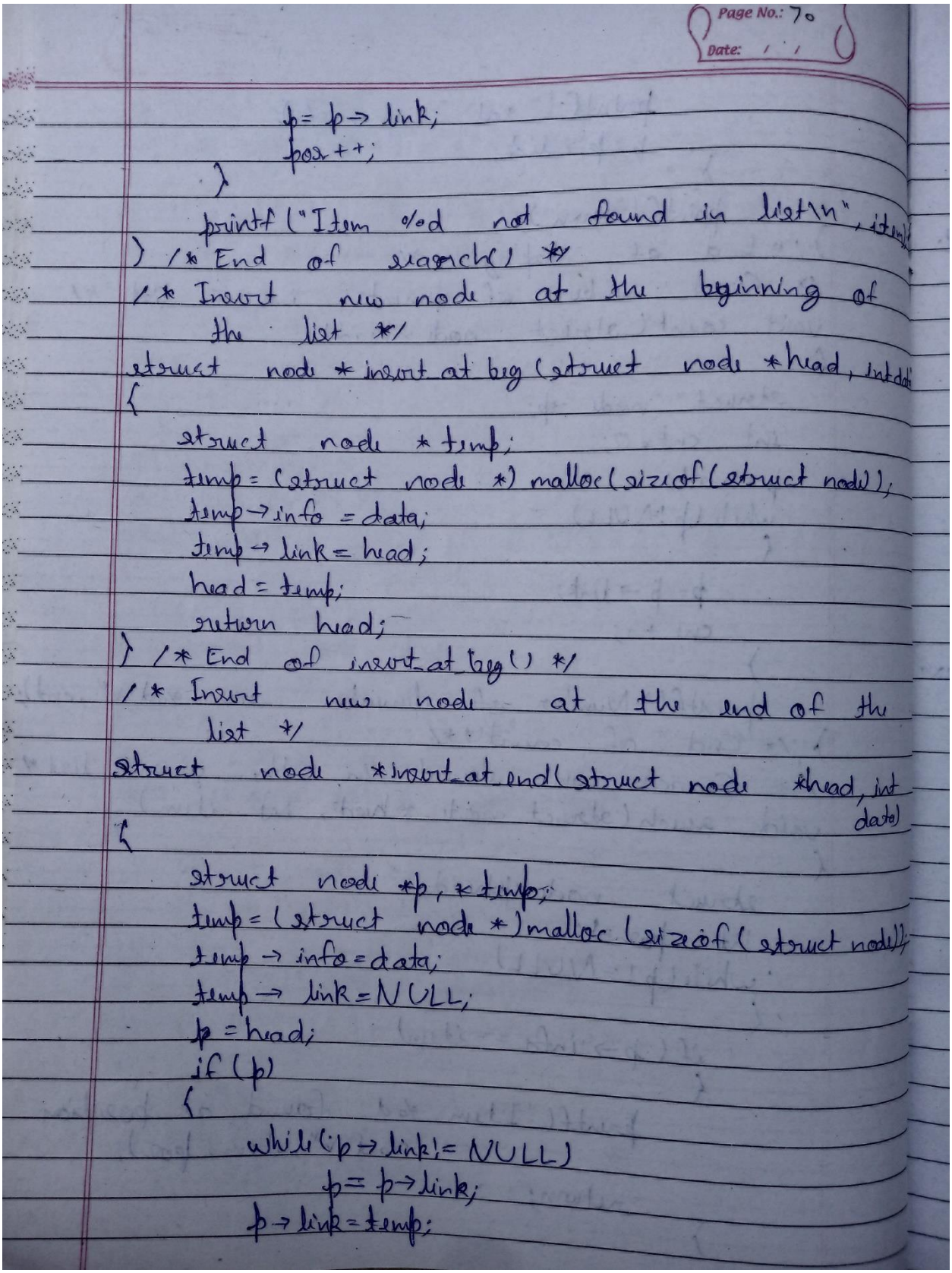


→ C Code -

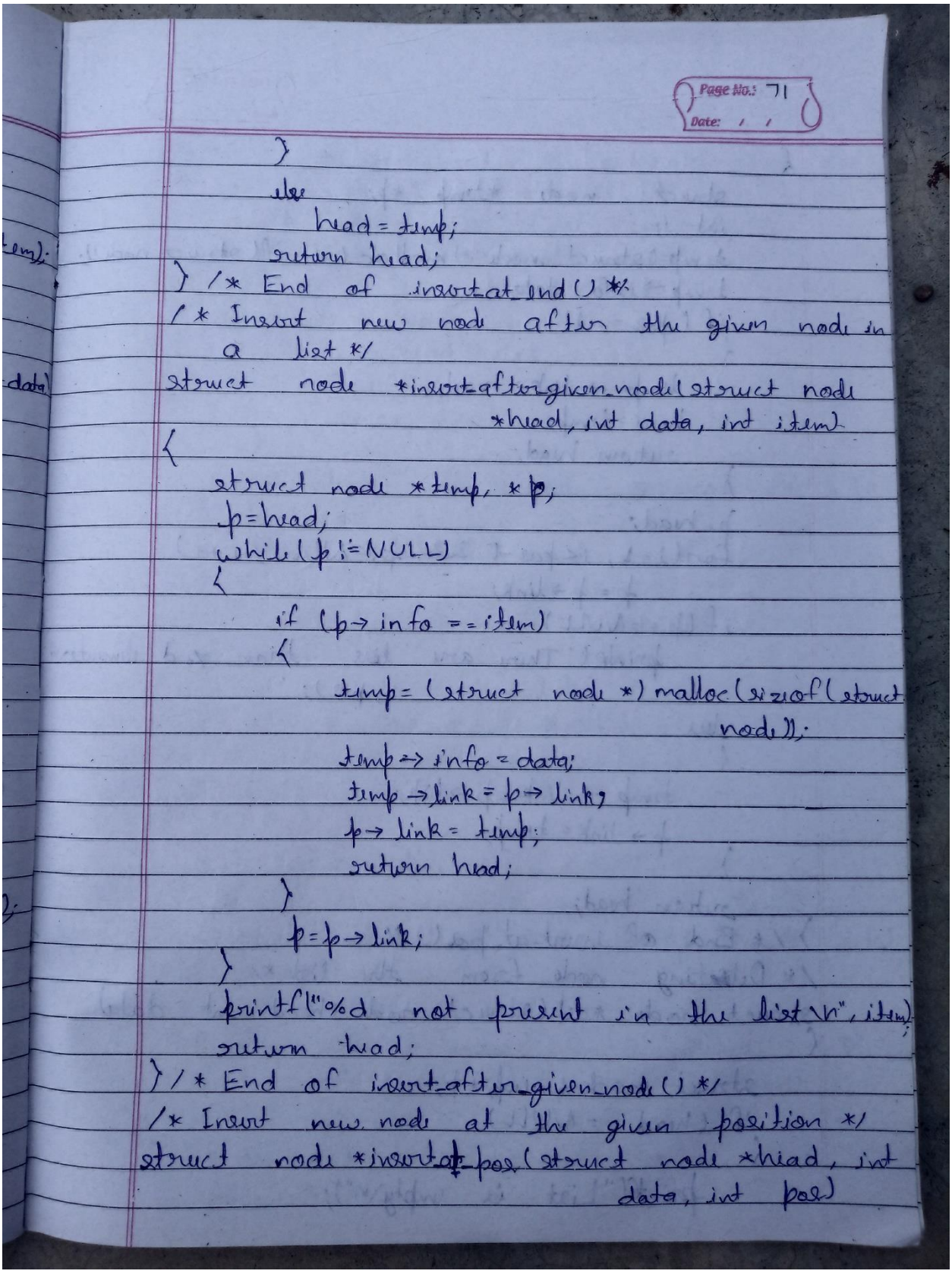
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int info;
    struct node *link;
};
/* Traverse the list or display every element
   in the list */
void display (struct node *head)
{
    struct node *p;
    if (head==NULL)
    {
        printf("List is empty\n");
        return;
    }
    p = head;
    printf("List is :\n");
    while(p!=NULL)
    {
```



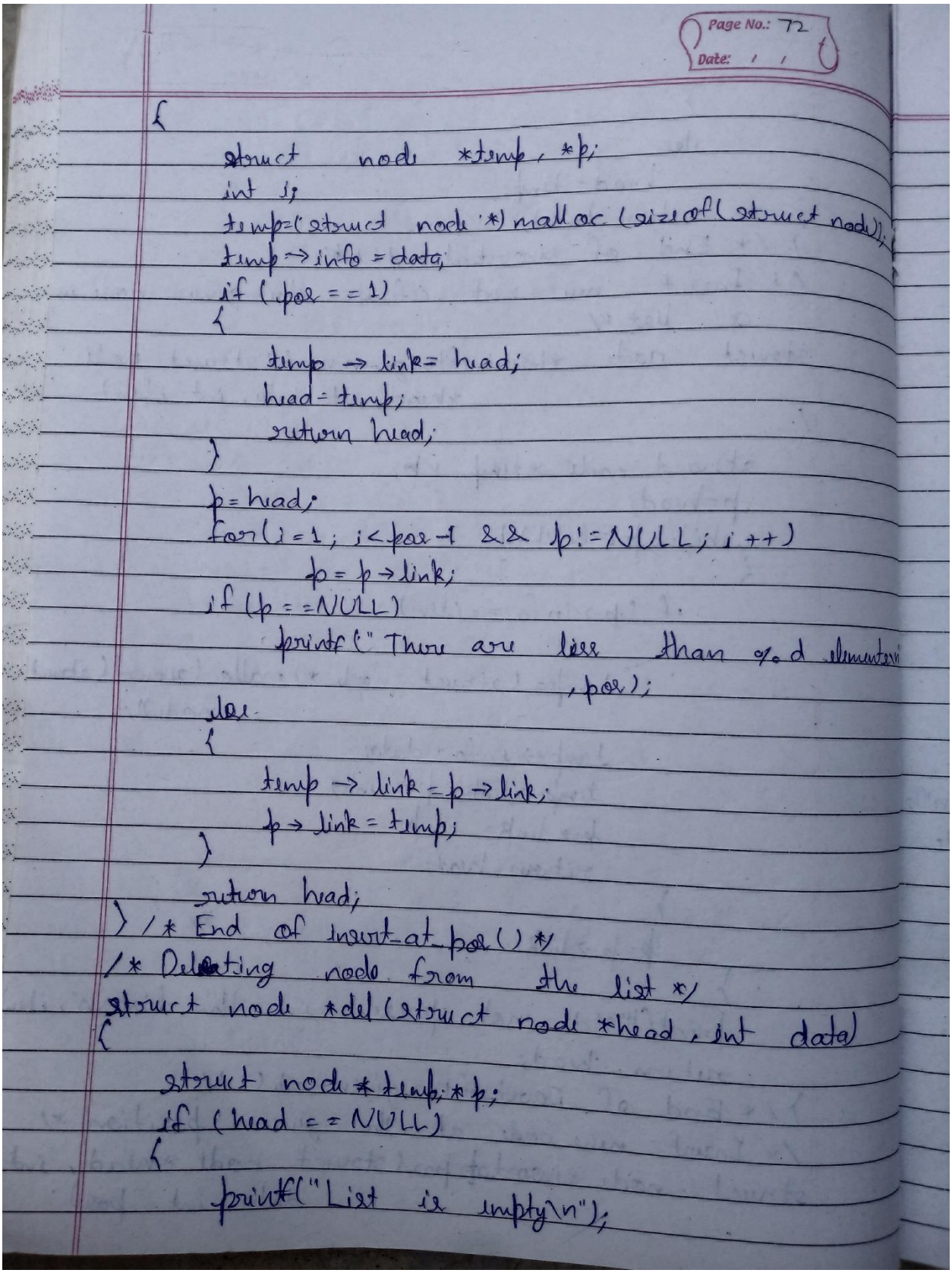
```
            printf(" %d ", p->info);
            p = p->link;
        }
        printf("\n\n");
}/* End of display() */
/* Count number of elements in a list */
void count(struct node *head)
{
    struct node *p;
    int cnt = 0;
    p = head;
    while(p != NULL)
    {
        p = p->link;
        cnt++;
    }
    printf("Number of elements are %d\n", cnt);
}/* End of count() */
/* Search an element in the given list */
void search(struct node *head, int item)
{
    struct node *p=head;
    int pos = 1;
    while(p != NULL)
    {
        if(p->info == item)
        {
            printf("Item %d found at position
                    %d\n", item, pos);
            return;
        }
```



```
            p = p -> link;
            pos++;
        }
        printf("Item %d not found in list\n", item);
} /* End of search() */
/* Insert new node at the beginning of
    the list */
struct node *insert_at_beg(struct node *head, int data)
{
    struct node *temp;
    temp = (struct node *) malloc(sizeof(struct node));
    temp->info = data;
    temp->link = head;
    head = temp;
    return head;
} /* End of insert_at_beg() */
/* Insert new node at the end of the
    list */
struct node *insert_at_end(struct node *head, int
                                        data)
{
    struct node *p, *temp;
    temp = (struct node *)malloc(sizeof(struct node));
    temp -> info = data;
    temp -> link = NULL;
    p = head;
    if(p)
    {
        while(p -> link != NULL)
            p = p -> link;
        p -> link = temp;
```

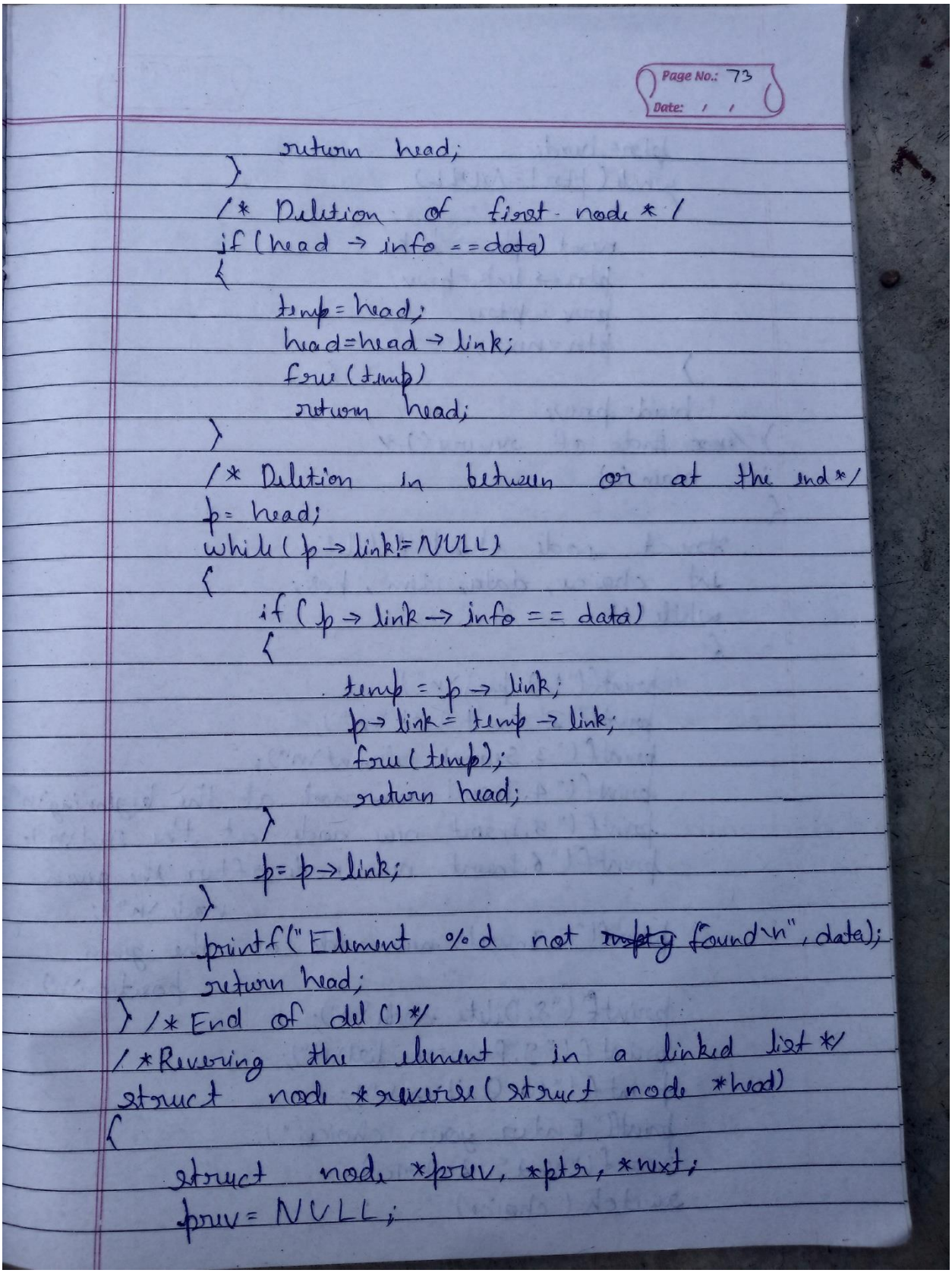


```
        }
        else
            head = temp;
        return head;
} /* End of insert_at_end() */
/* Insert new node after the given node in
   a list */
struct node *insert_after_given_node(struct node
                    *head, int data, int item)
{
    struct node *temp, *p;
    p=head;
    while(p != NULL)
    {
        if (p->info == item)
        {
            temp = (struct node *) malloc(sizeof(struct
                                        node));
            temp->info = data;
            temp->link = p->link;
            p->link = temp;
            return head;
        }
        p = p->link;
    }
    printf("%d not present in the list\n", item);
    return head;
}/* End of insert_after_given_node() */
/* Insert new node at the given position */
struct node *insert_at_pos(struct node *head, int
                    data, int pos)
```

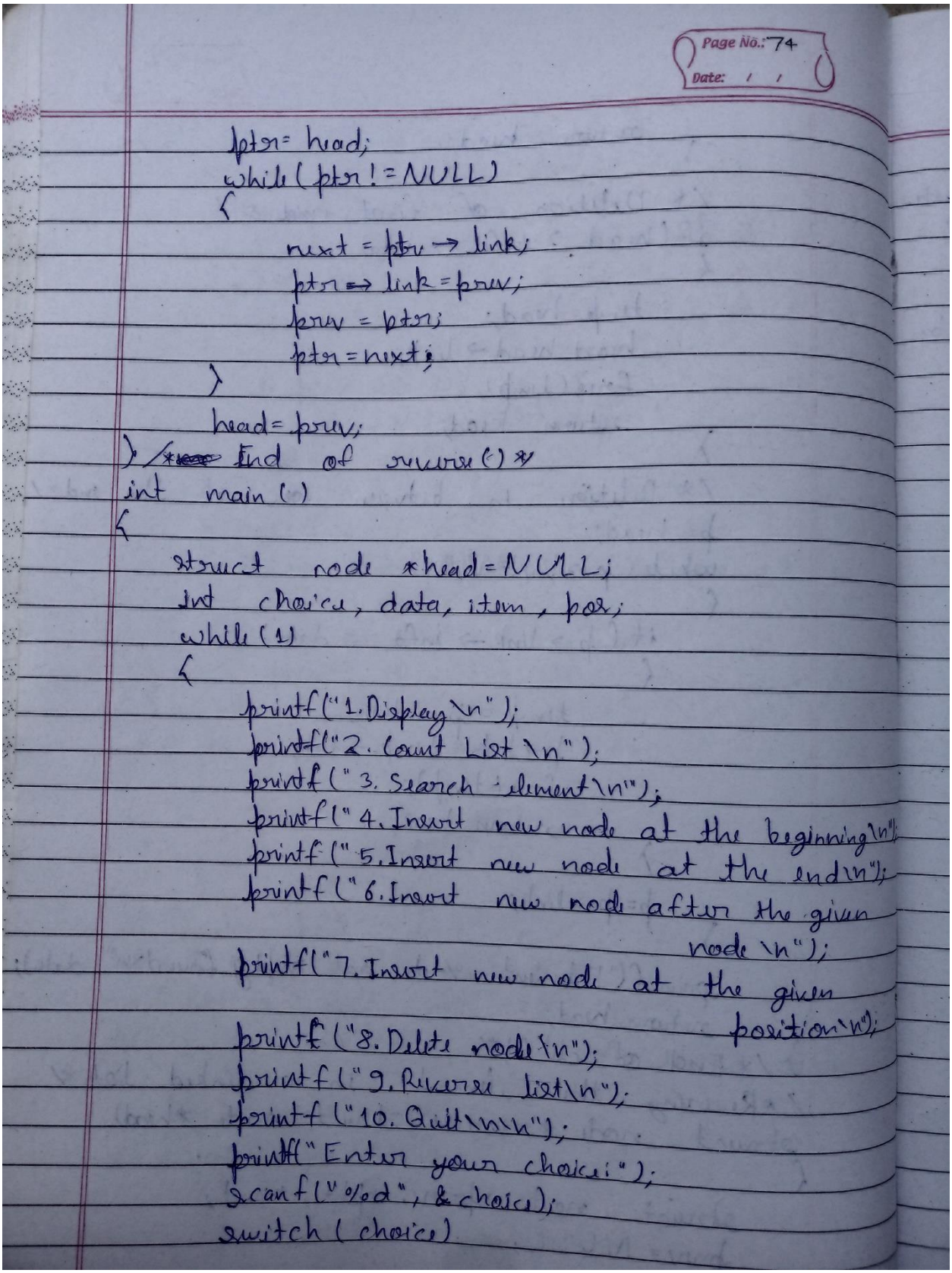


Page No.: 72
Date: / /

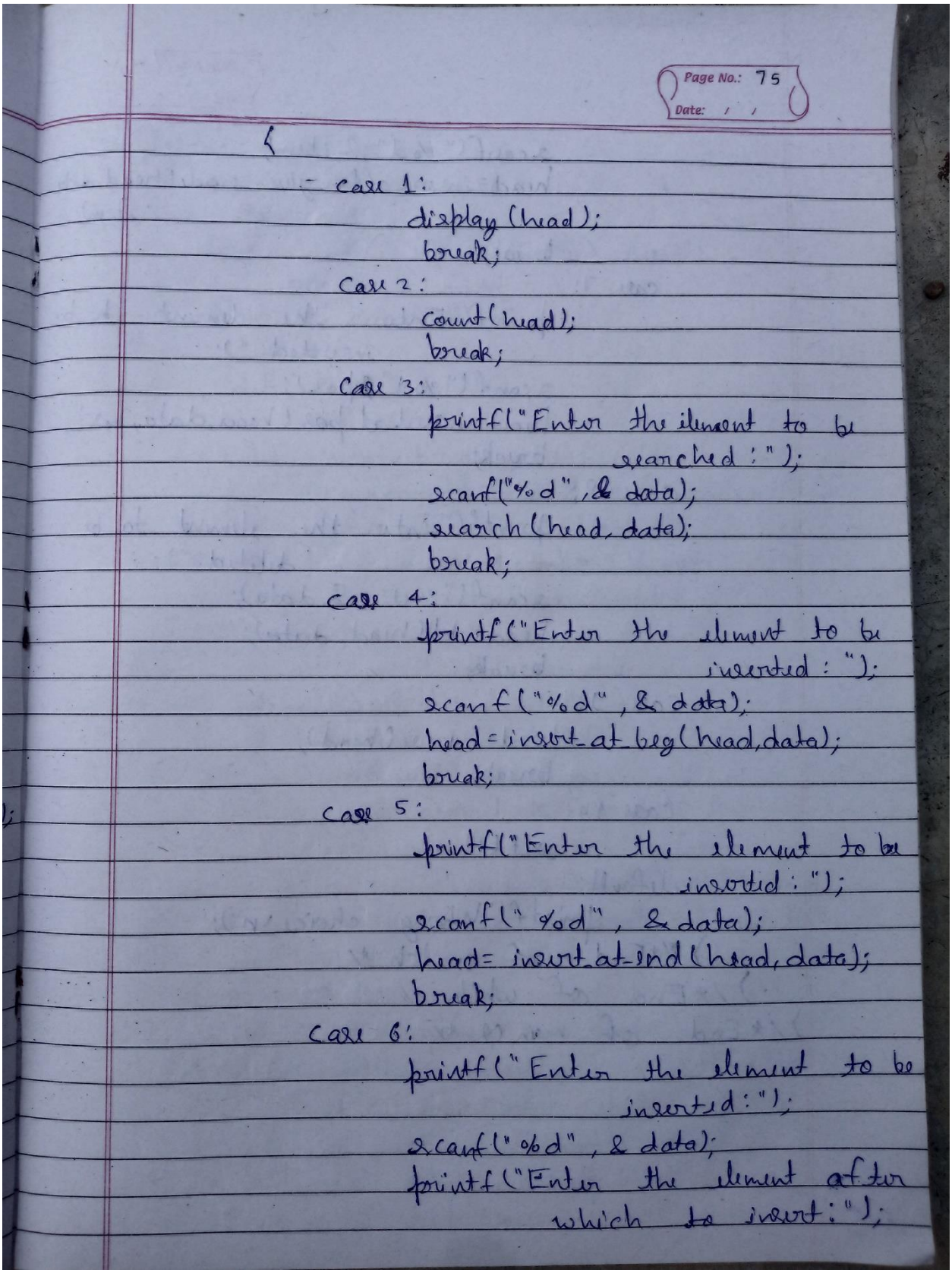
```
{
    struct node *temp, *p;
    int i;
    temp=(struct node *) malloc (sizeof(struct node));
    temp -> info = data;
    if (pos == 1)
    {
        temp -> link= head;
        head=temp;
        return head;
    }
    p=head;
    for(i=1; i<pos-1 && p!=NULL; i++)
        p=p->link;
    if (p==NULL)
        printf(" There are less than %d elements"
                ,pos);
    else
    {
        temp -> link=p->link;
        p->link=temp;
    }
    return head;
} /* End of insert_at_pos() */
/* Deleting node from the list */
struct node *del(struct node *head, int data)
{
    struct node *temp, *p;
    if (head == NULL)
    {
        printf("List is empty\n");
```



```
        return head;
    }
    /* Dulition of first node */
    if (head -> info == data)
    {
        temp = head;
        head = head -> link;
        free (temp)
        return head;
    }
    /* Dulition in between or at the end*/
    p = head;
    while (p -> link != NULL)
    {
        if (p -> link -> info == data)
        {
            temp = p -> link;
            p -> link = temp -> link;
            free (temp);
            return head;
        }
        p = p -> link;
    }
    printf("Element %d not ~~mpty~~ found\n", data);
    return head;
} /* End of del() */
/* Reversing the element in a linked list */
struct node *reverse (struct node *head)
{
    struct node *prev, *ptr, *nxt;
    prev = NULL;
```

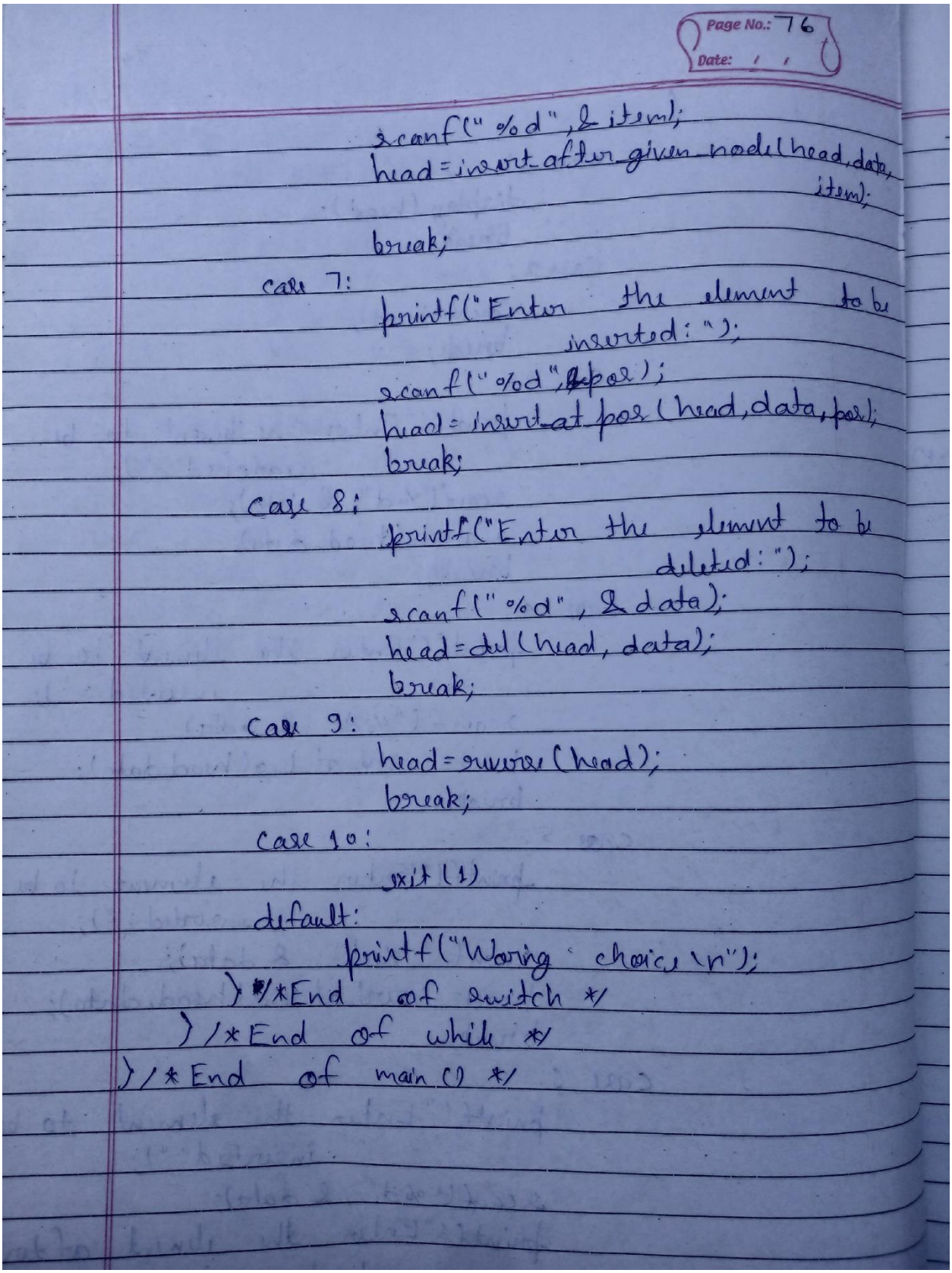


```
        ptr = head;
        while (ptr != NULL)
        {
            next = ptr -> link;
            ptr -> link = prev;
            prev = ptr;
            ptr = next;
        }
        head = prev;
}  /* End of reverse() */
int main()
{
    struct node *head = NULL;
    int choice, data, item, pos;
    while (1)
    {
        printf("1. Display \n");
        printf("2. Count List \n");
        printf("3. Search element \n");
        printf("4. Insert new node at the beginning \n");
        printf("5. Insert new node at the end \n");
        printf("6. Insert new node after the given
                                        node \n");
        printf("7. Insert new node at the given
                                        position \n");
        printf("8. Delete node \n");
        printf("9. Reverse list \n");
        printf("10. Quit \n\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
```

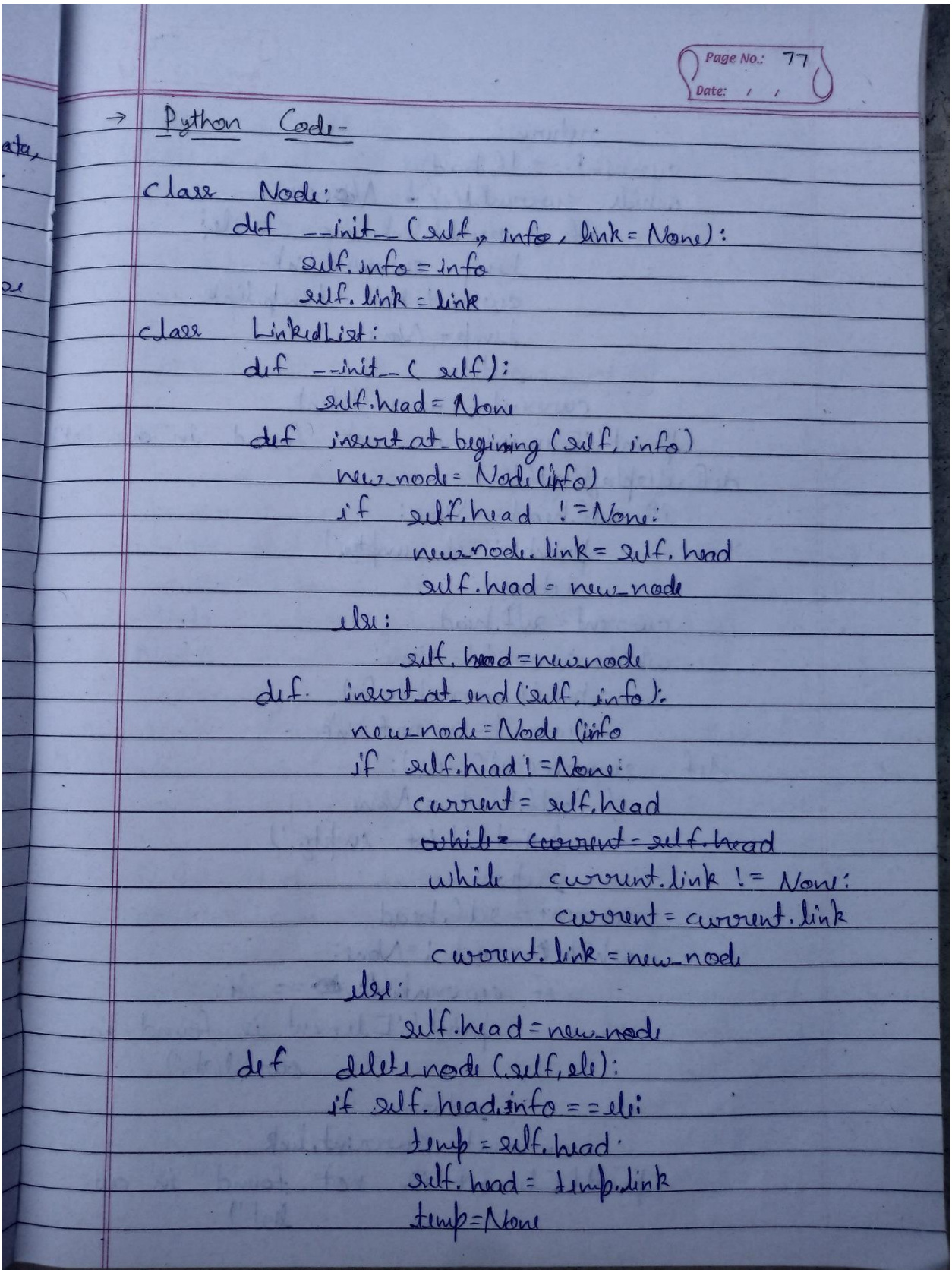


Page No.: 75
Date: / /

```
{
    case 1:
        display(head);
        break;
    case 2:
        count(head);
        break;
    case 3:
        printf("Enter the element to be
                searched:");
        scanf("%d", &data);
        search(head, data);
        break;
    case 4:
        printf("Enter the element to be
                inserted:");
        scanf("%d", &data);
        head = insert_at_beg(head, data);
        break;
    case 5:
        printf("Enter the element to be
                inserted:");
        scanf("%d", &data);
        head = insert_at_end(head, data);
        break;
    case 6:
        printf("Enter the element to be
                inserted:");
        scanf("%d", &data);
        printf("Enter the element after
                which to insert:");
```



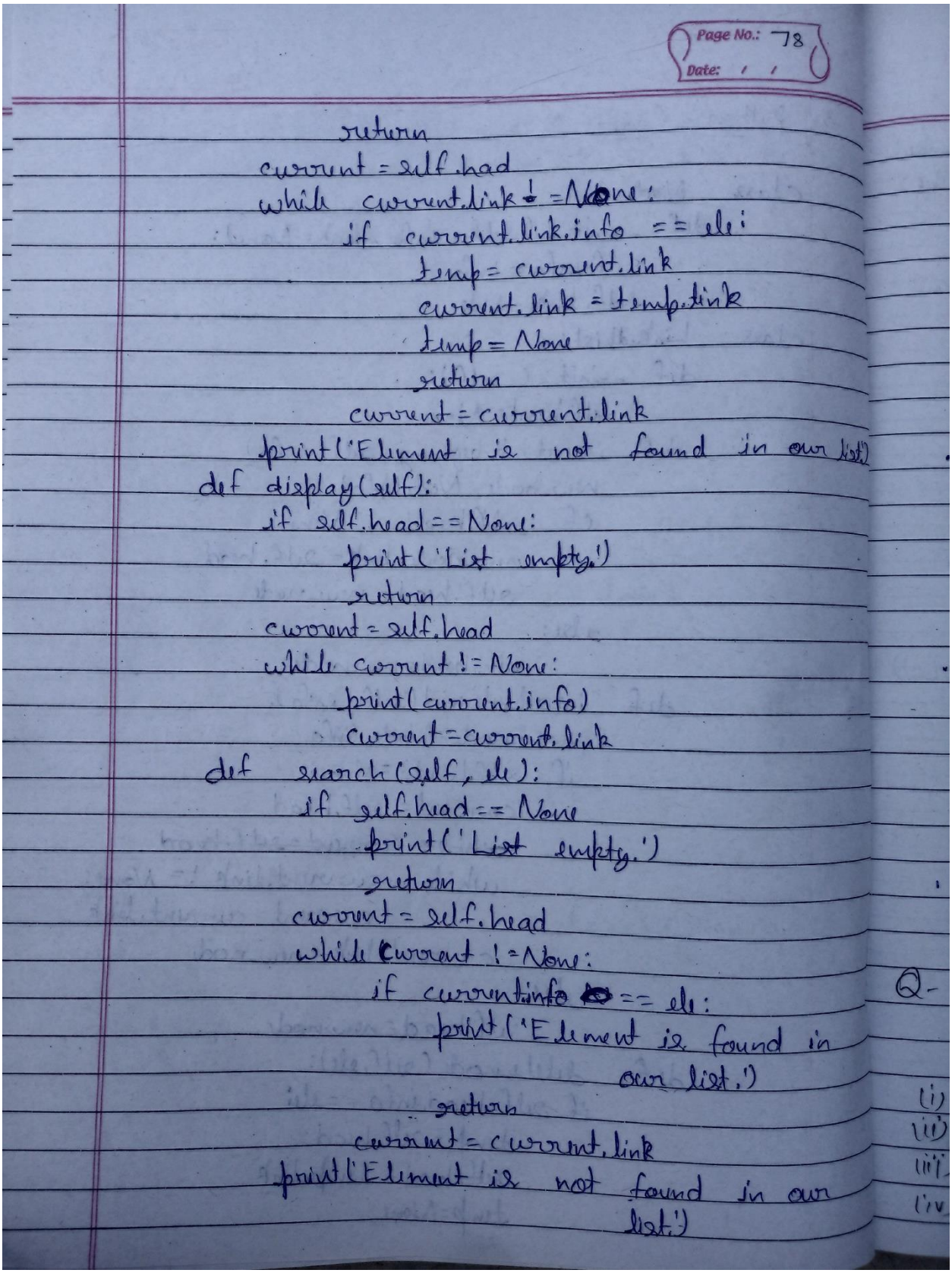
```
                scanf("%d", &item);
                head = insert_after_given_node(head, data,
                                               item);
                break;
        case 7:
                printf("Enter the element to be
                        inserted: ");
                scanf("%d", &pos);
                head = insert_at_pos(head, data, pos);
                break;
        case 8:
                printf("Enter the element to be
                                deleted: ");
                scanf("%d", &data);
                head = del(head, data);
                break;
        case 9:
                head = reverse(head);
                break;
        case 10:
                exit(1)
        default:
                printf("Wrong choice\n");
    }/*End of switch */
  }/*End of while */
}/*End of main() */
```



Page No.: 77
Date: / /

→ Python Code-

```
class Node:
    def __init__(self, info, link=None):
        self.info = info
        self.link = link
class LinkedList:
    def __init__(self):
        self.head = None
    def insert_at_beginning(self, info)
        new_node = Node(info)
        if self.head != None:
            new_node.link = self.head
            self.head = new_node
        else:
            self.head = new_node
    def insert_at_end(self, info):
        new_node = Node(info
        if self.head != None:
            current = self.head
            ~~while current = self.head~~
            while current.link != None:
                current = current.link
            current.link = new_node
        else:
            self.head = new_node
    def delete_node(self, ele):
        if self.head.info == ele:
            temp = self.head
            self.head = temp.link
            temp = None
```



```
                return
        current = self.head
        while current.link != None:
            if current.link.info == ele:
                temp = current.link
                current.link = temp.link
                temp = None
                return
            current = current.link
        print('Element is not found in our list')
    def display(self):
        if self.head == None:
            print('List empty.')
            return
        current = self.head
        while current != None:
            print(current.info)
            current = current.link
    def search(self, ele):
        if self.head == None
            print('List empty.')
            return
        current = self.head
        while current != None:
            if current.info == ele:
                print('Element is found in
                        our list.')
                return
            current = current.link
        print('Element is not found in our
                        list.')
```

Q-

(i)
(ii)
(iii)
(iv)

Page No.: 79
Date: / /

```
LL = LinkedList()
LL.insert_at_beginning(10)
LL.insert_at_end(5)
LL.delete_node(10)
LL.search(10)
LL.display()
```

Drawbacks of Singly Linked List -

- They use more memory than arrays because of the storage used by their pointers.
- Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access.
- Nodes are stored incontiguously, greatly increasing the time periods required to access individual elements within the lists, ~~often it comes to reverse traversing. For example~~ especially with a CPU cache.
- Difficulties arise in linked lists when it comes to reverse traversing.

Q- In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is -

(i) $\log n$
(ii) $n/2$
(iii) $\log n - 1$.
(iv) n (✓)

Page No.: 80
Date: / /

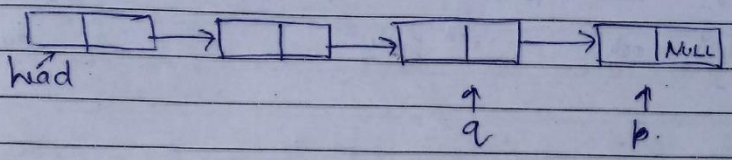
Q- The following C function take a singly-linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is back blank-

```
type struct node
{
    int value;
    struct node *next;
} Node;
Node *move_to_front (Node *head)
{
    Node *p, *q;
    if((head == NULL) || (head->next == NULL))
        return head;
    q = NULL; p = head;
    while (p->next != NULL)
    {
        q = p;
        p = p->next;
    }
    _____
    _____
    return head;
}
```

Choose the correct alternatives to replace the black line-

Page No.: 81
Date: / /

(i) q = NULL ; p → next = head ; head = p ;
(ii) q → next = NULL ; head = p ; p → next = head ;
(iii) head = p ; p → next = q ; q → next = NULL ;
(iv) q → next = NULL ; p → next = head ; head = p ; (✓)



q → next = NULL
p → next = head
head = p

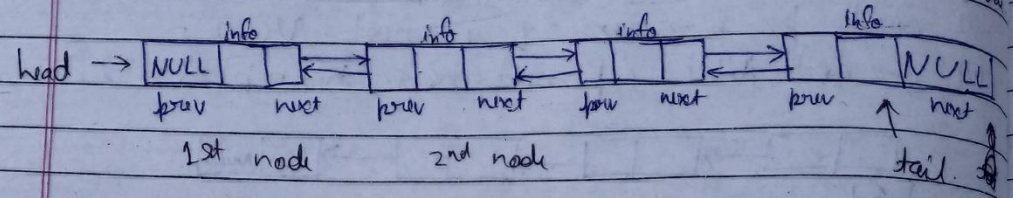
Note -

- In Singly linked list we do not traverse list in reverse order but in case of doubly linked list we have link back to the previous node we traverse easyly the list in reverse order.

- In case of singly linked list if we want to insert a node before a node then we only have a choice to traverse the list and find the previous and next node link which is done in O(n) complexity but in case of doubly linked list we have already the previous and next node link for every node so we donot traverse the list and it done in O(1) complexity.

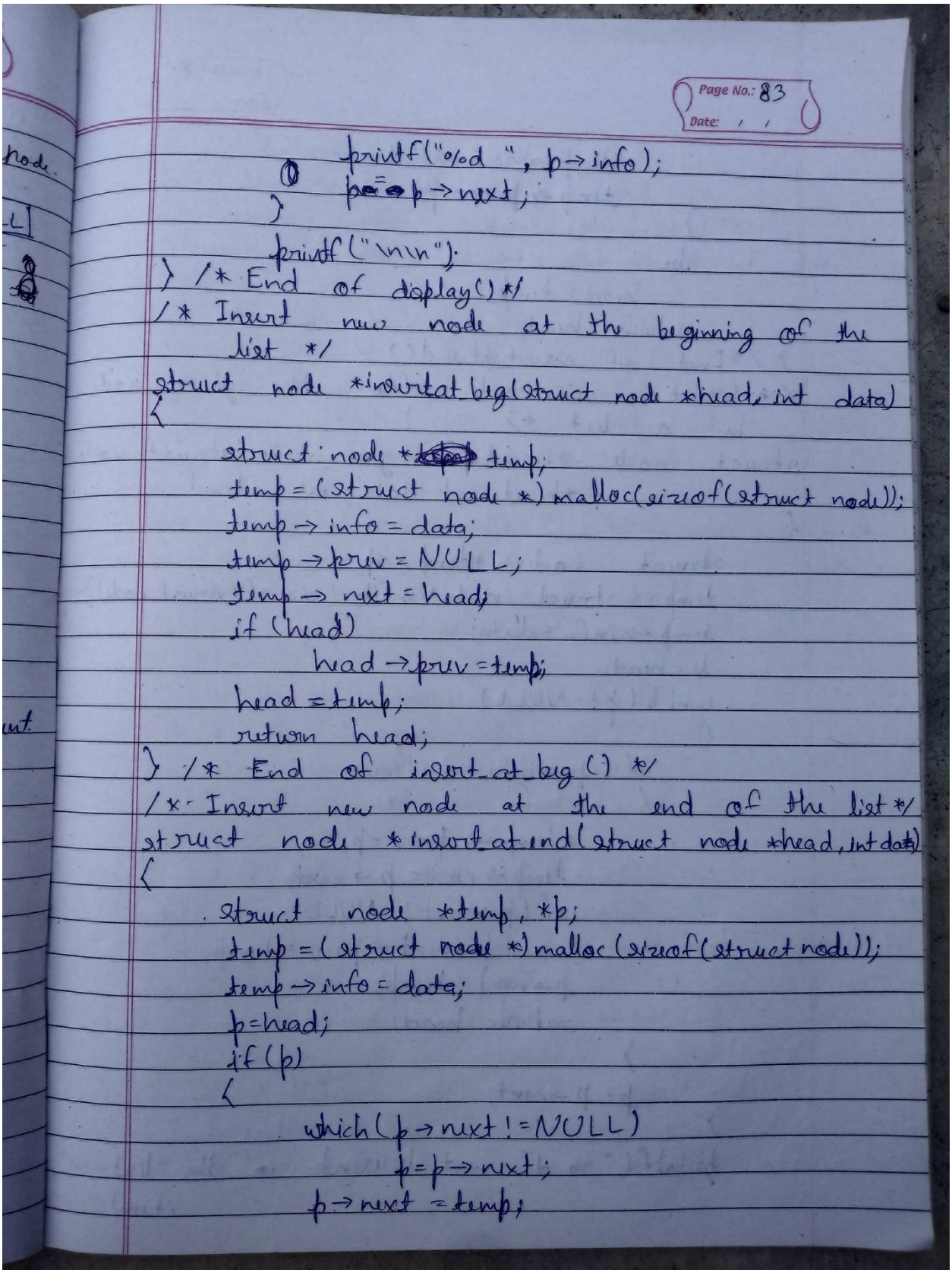
Page No.: 82
Date: / /

Doubly Linked List-

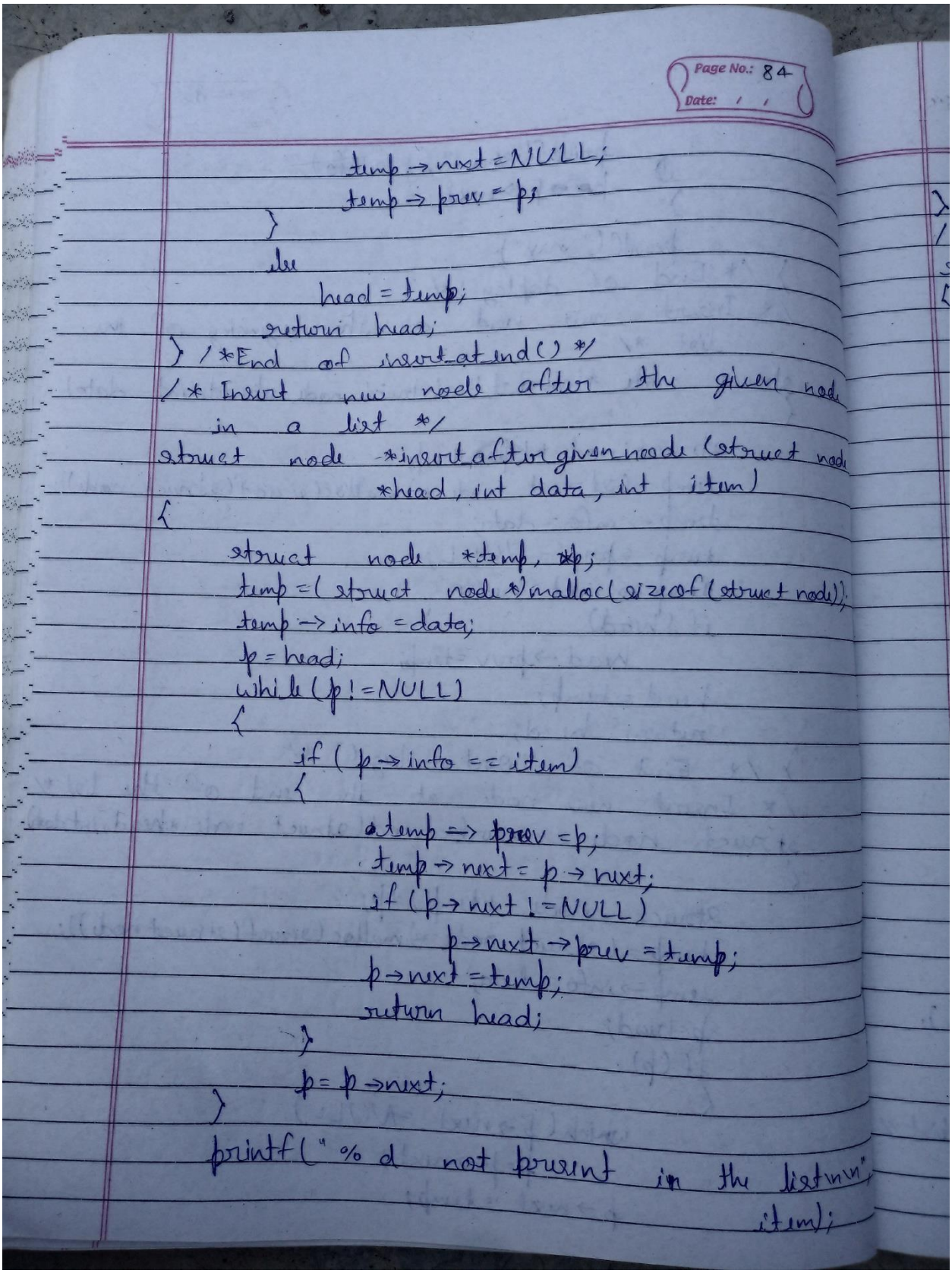


C Code-

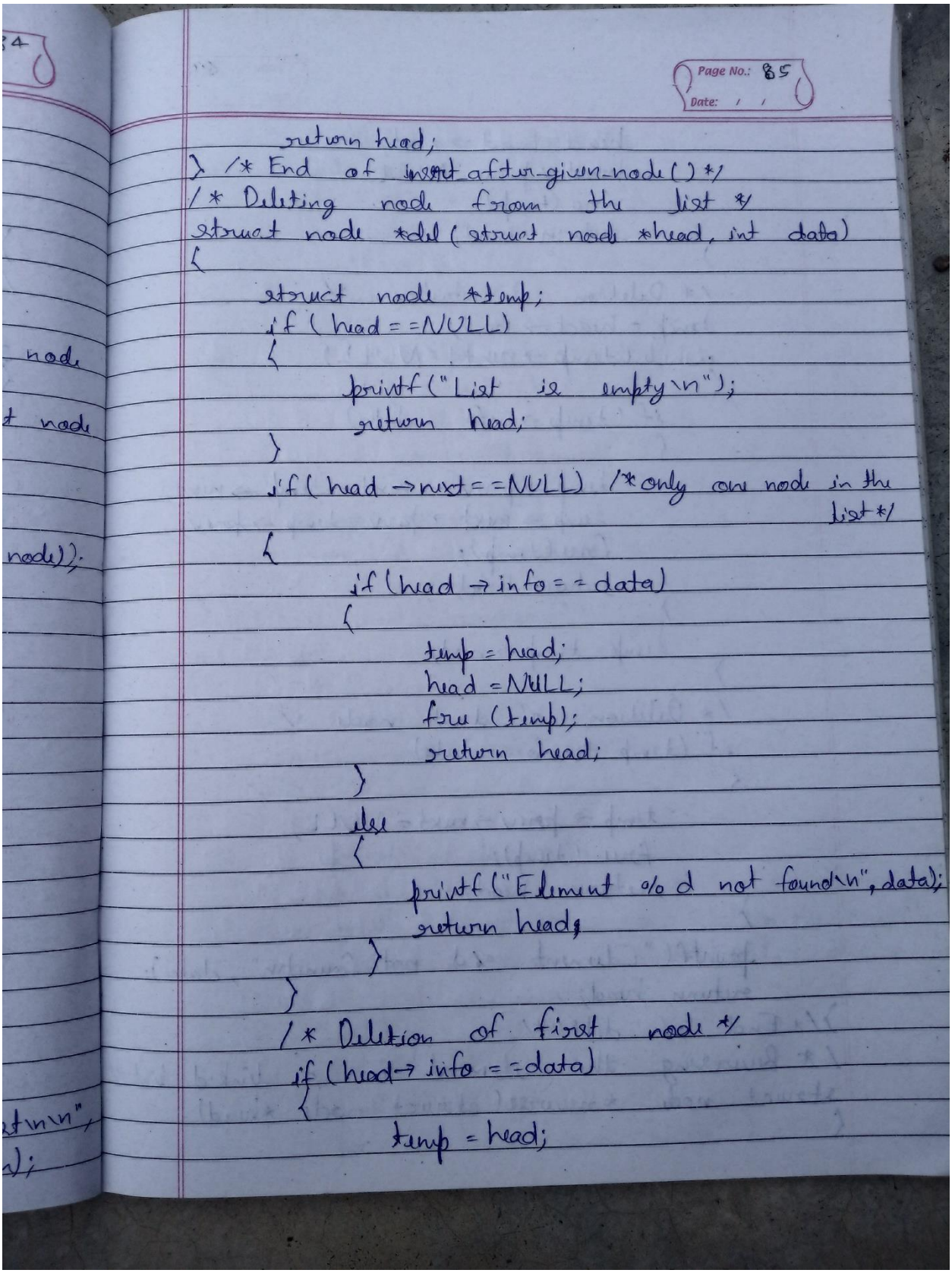
```
#include<stdio.h>
#include<conio.h>
struct node
{
    struct node *prev;
    int info;
    struct node *next;
};
/* Traverse the list or display every element
   in the list */
void display(struct node *head)
{
    struct node *p;
    if(head==NULL)
    {
        printf("List is empty\n\n");
        return;
    }
    p=head;
    printf("List is:\t");
    while(p!=NULL)
    {
```



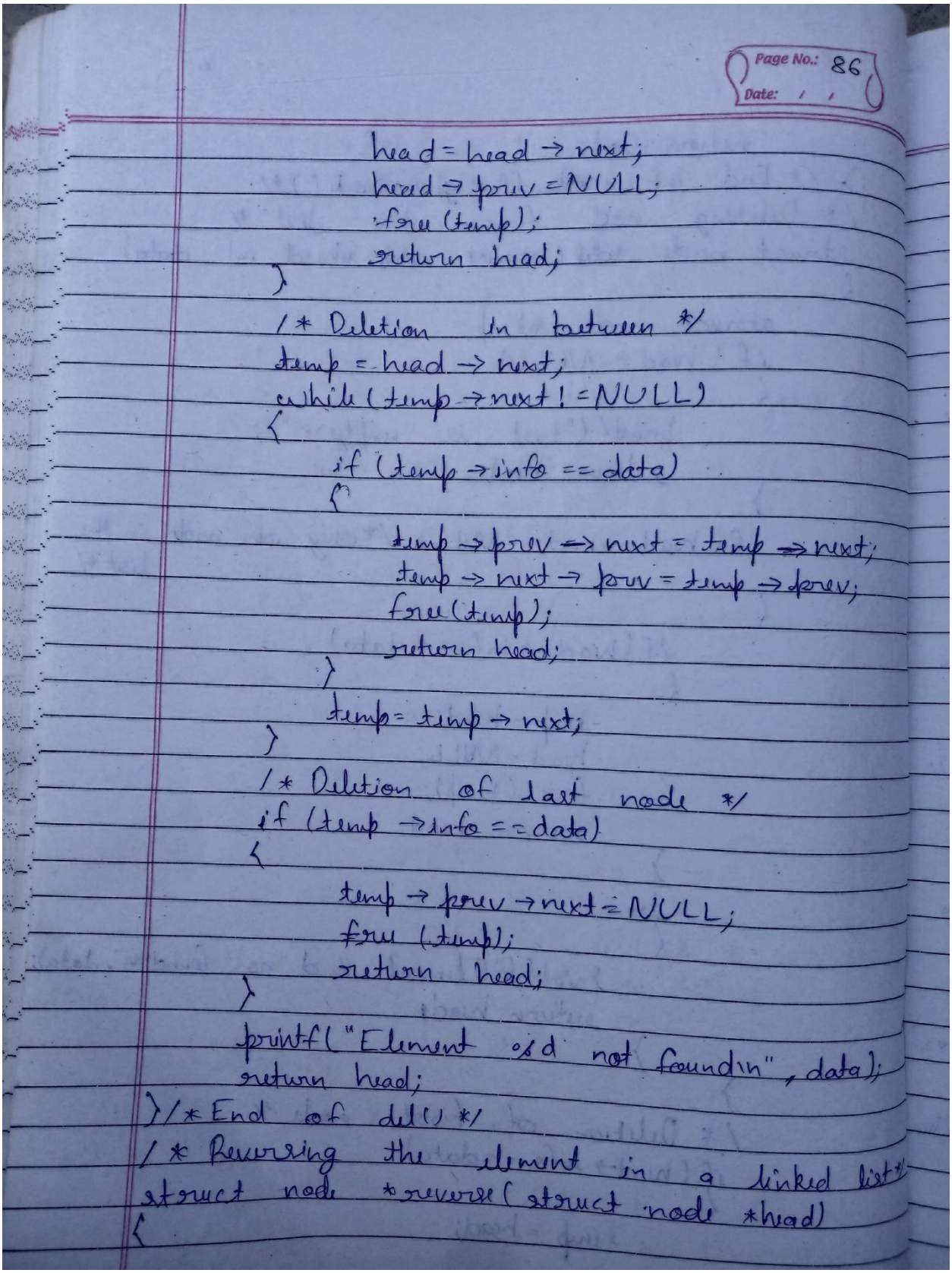
```
            printf("%d ", p->info);
            p = p->next;
        }
        printf("\n\n");
} /* End of display() */
/* Insert new node at the beginning of the
   list */
struct node *insrtat_beg(struct node *head, int data)
{
    struct node *temp;
    temp = (struct node *) malloc(sizeof(struct node));
    temp->info = data;
    temp->prev = NULL;
    temp->next = head;
    if (head)
        head->prev = temp;
    head = temp;
    return head;
} /* End of insert_at_beg() */
/* Insert new node at the end of the list */
struct node *insrt_at_end(struct node *head, int data)
{
    struct node *temp, *p;
    temp = (struct node *) malloc(sizeof(struct node));
    temp->info = data;
    p = head;
    if (p)
    {
        while(p->next != NULL)
            p = p->next;
        p->next = temp;
```



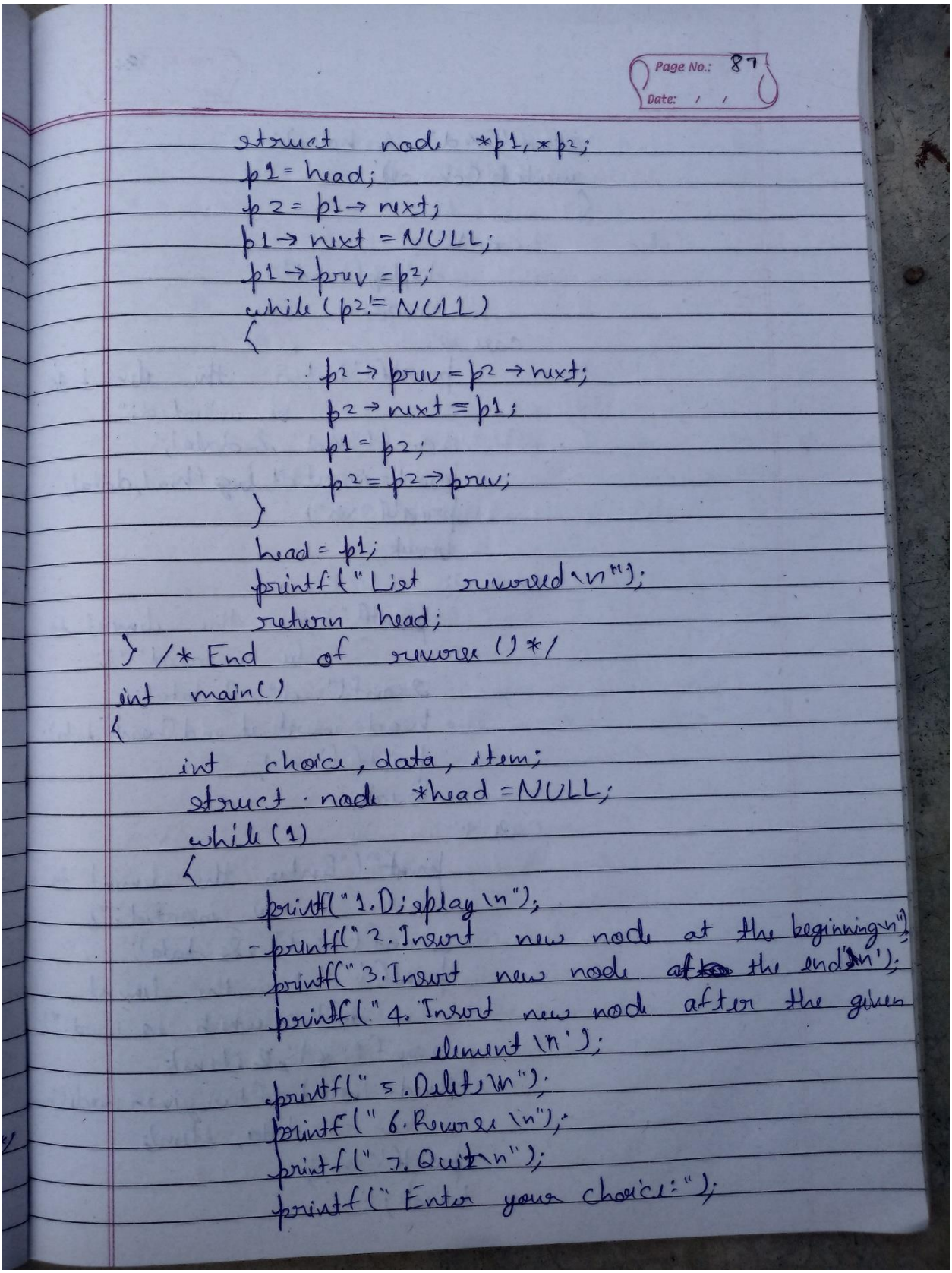
```
            temp -> next = NULL;
            temp -> prev = p;
        }
        else
            head = temp;
        return head;
} /*End of insert_at_end() */
/* Insert new node after the given node
   in a list */
struct node *insert_after_given_node (struct node
                *head, int data, int item)
{
    struct node *temp, *p;
    temp = (struct node *)malloc(sizeof(struct node));
    temp -> info = data;
    p = head;
    while (p != NULL)
    {
        if ( p -> info == item)
        {
            temp -> prev = p;
            temp -> next = p -> next;
            if (p -> next != NULL)
                p -> next -> prev = temp;
            p -> next = temp;
            return head;
        }
        p = p -> next;
    }
    printf("%d not present in the list\n",
                                    item);
```



```
Page No.: 85
Date:

    return head;
} /* End of insert_after_given_node() */
/* Deleting node from the list */
struct node *del(struct node *head, int data)
{
    struct node *temp;
    if(head==NULL)
    {
        printf("List is empty\n");
        return head;
    }
    if(head->next==NULL) /*only one node in the
                           list*/
    {
        if(head->info==data)
        {
            temp=head;
            head=NULL;
            free(temp);
            return head;
        }
        else
        {
            printf("Element %d not found\n",data);
            return head;
        }
    }
    /* Deletion of first node */
    if(head->info==data)
    {
        temp=head;
```

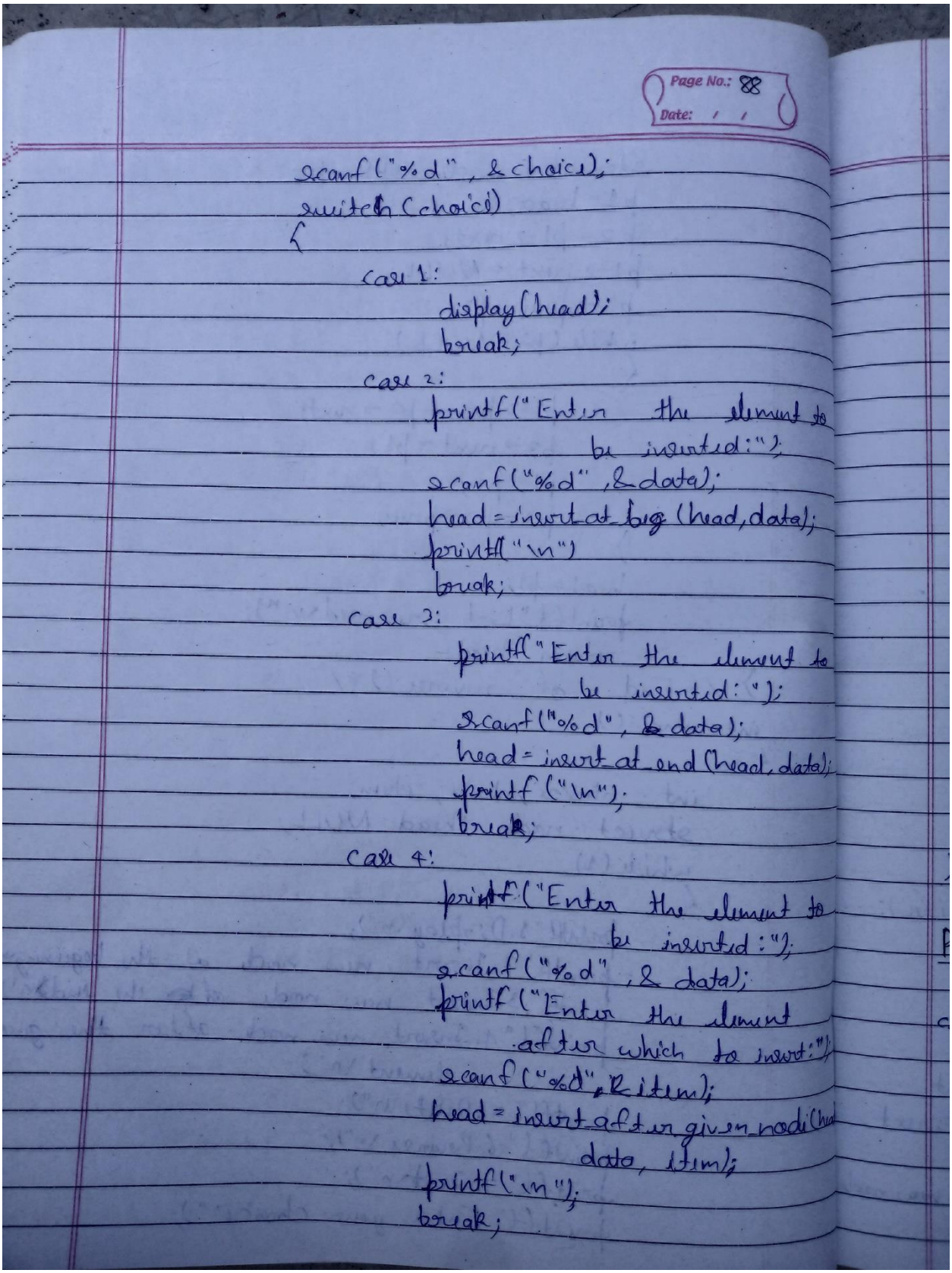


```
Page No.: 86
Date: / /

        head = head -> next;
        head -> prev = NULL;
        free (temp);
        return head;
    }
    /* Deletion in between */
    temp = head -> next;
    while (temp -> next != NULL)
    {
        if (temp -> info == data)
        {
            temp -> prev -> next = temp -> next;
            temp -> next -> prev = temp -> prev;
            free (temp);
            return head;
        }
        temp = temp -> next;
    }
    /* Deletion of last node */
    if (temp -> info == data)
    {
        temp -> prev -> next = NULL;
        free (temp);
        return head;
    }
    printf ("Element %d not found\n", data);
    return head;
}/* End of del() */
/* Reversing the element in a linked list */
struct node *reverse (struct node *head)
{
```

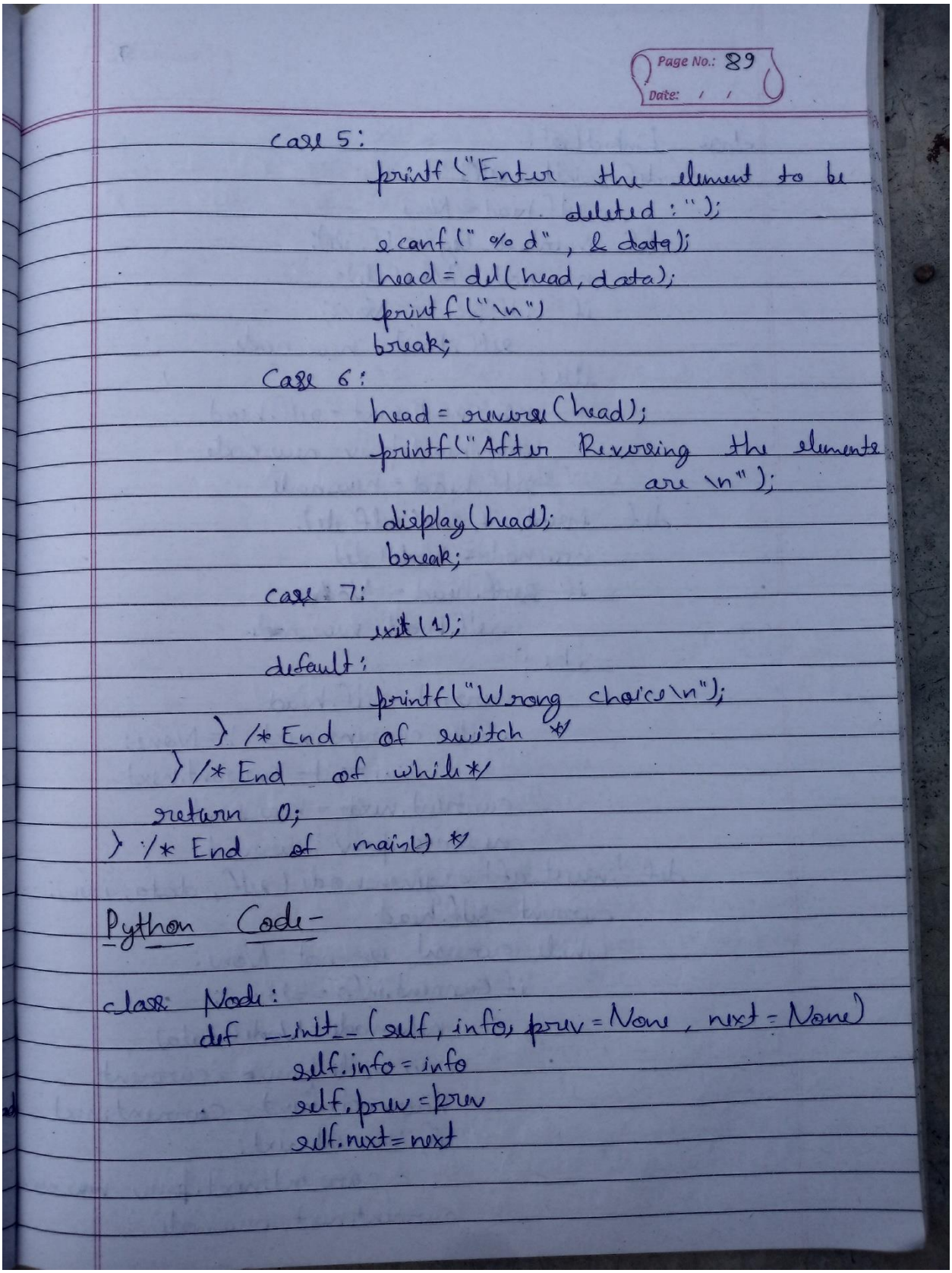


Page No.: 87
Date: / /

```
struct node *p1, *p2;
p1 = head;
p2 = p1->next;
p1->next = NULL;
p1->prev = p2;
while (p2 != NULL)
{
    p2->prev = p2->next;
    p2->next = p1;
    p1 = p2;
    p2 = p2->prev;
}
head = p1;
printf("List reversed\n");
return head;
} /* End of reverse() */
int main()
{
    int choice, data, item;
    struct node *head = NULL;
    while (1)
    {
        printf("1. Display\n");
        printf("2. Insert new node at the beginning\n");
        printf("3. Insert new node at the end\n");
        printf("4. Insert new node after the given
                element\n");
        printf("5. Delete\n");
        printf("6. Reverse\n");
        printf("7. Quit\n");
        printf("Enter your choice:");
```



```
scanf("%d", &choice);
switch(choice)
{
    case 1:
        display(head);
        break;
    case 2:
        printf("Enter the element to
                be inserted:");
        scanf("%d", &data);
        head = insert_at_beg(head, data);
        printf("\n")
        break;
    case 3:
        printf("Enter the element to
                be inserted:");
        scanf("%d", &data);
        head = insert_at_end(head, data);
        printf("\n");
        break;
    case 4:
        printf("Enter the element to
                be inserted:");
        scanf("%d", &data);
        printf("Enter the element
                after which to insert:");
        scanf("%d", &item);
        head = insert_after_given_node(head,
                data, item);
        printf("\n");
        break;
```



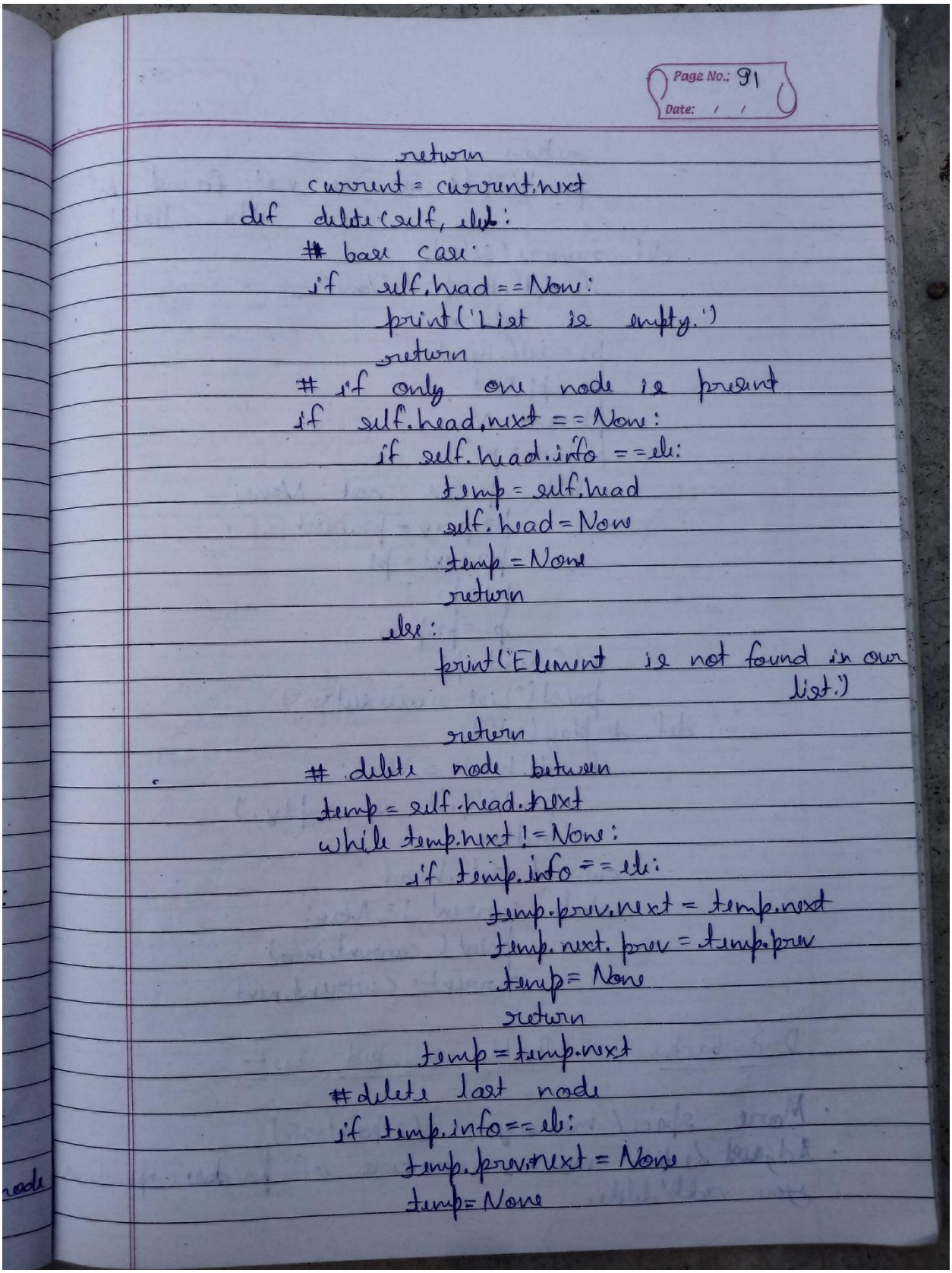
```
Page No.: 89
Date: / /

case 5:
        printf("Enter the element to be
                deleted :");
        scanf("%d", &data);
        head = del(head, data);
        printf("\n")
        break;
case 6:
        head = reverse(head);
        printf("After Reversing the elements
                are \n");
        display(head);
        break;
case 7:
        exit(1);
default:
        printf("Wrong choice\n");
    } /* End of switch */
  } /* End of while */
  return 0;
} /* End of main() */
```

Python Code-

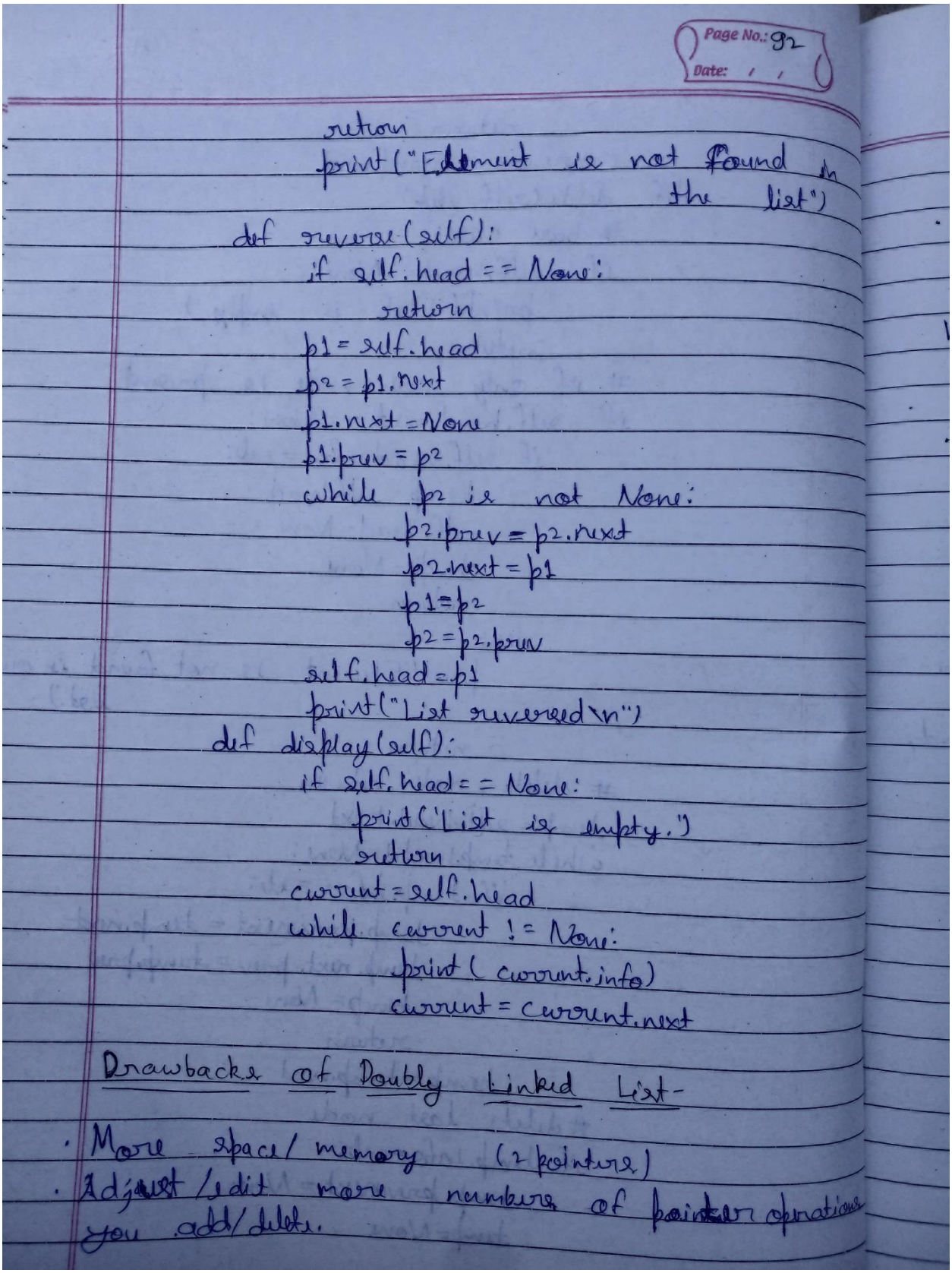
```
class Node:
    def __init__(self, info, prev=None, next=None)
        self.info = info
        self.prev = prev
        self.next = next
```

Page No.: 90
Date: / /

```
class LinkedList:
    def __init__(self):
        self.head = None
    def insert_at_beg(self, ele):
        new_node = None(ele).
        if self.head == None:
            self.head = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
    def insert_at_end(self, ele):
        new_node = Node(ele)
        if self.head == None:
            self.head = new_node
        else:
            current = self.head
            while current.next != None:
                current = current.next
            current.next = new_node
            new_node.prev = current
def insert_after_given_node(self, data, item):
    current = self.head
    while current is not None:
        if Current.info == item:
            new_node = Node(data)
            new_node.prev = current
            new_node.next = Current.next
            if current.next:
                current.next.prev = new_node
            current.next = new_node
```



```
                return
            current = current.next
    def delete(self, ele):
        # base case
        if self.head == None:
            print('List is empty.')
            return
        # if only one node is present
        if self.head.next == None:
            if self.head.info == ele:
                temp = self.head
                self.head = None
                temp = None
                return
            else:
                print('Element is not found in our
                                        list.')

                return
        # delete node between
        temp = self.head.next
        while temp.next != None:
            if temp.info == ele:
                temp.prev.next = temp.next
                temp.next.prev = temp.prev
                temp = None
                return
            temp = temp.next
        #delete last node
        if temp.info == ele:
            temp.prev.next = None
            temp = None
```



Page No.: 92
Date: / /

```
            return
        print("Element is not found in
                                the list")
    def reverse(self):
        if self.head == None:
            return
        p1 = self.head
        p2 = p1.next
        p1.next = None
        p1.prev = p2
        while p2 is not None:
            p2.prev = p2.next
            p2.next = p1
            p1 = p2
            p2 = p2.prev
        self.head = p1
        print("List reversed\n")
    def display(self):
        if self.head == None:
            print('List is empty.')
            return
        current = self.head
        while current != None:
            print(current.info)
            current = current.next
```

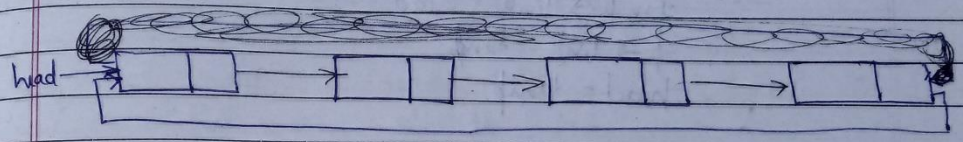
Drawbacks of Doubly Linked List-

- More space/ memory (2 pointers)
- Adjust/edit more numbers of pointer operations you add/delete.

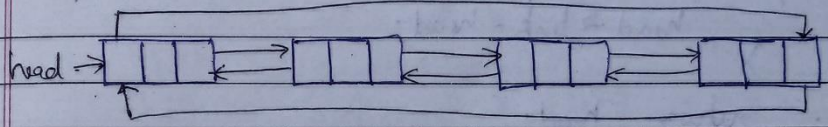
Page No.: 93
Date: 31/10/20

Circular Linked List -

Circular Singly Linked List -

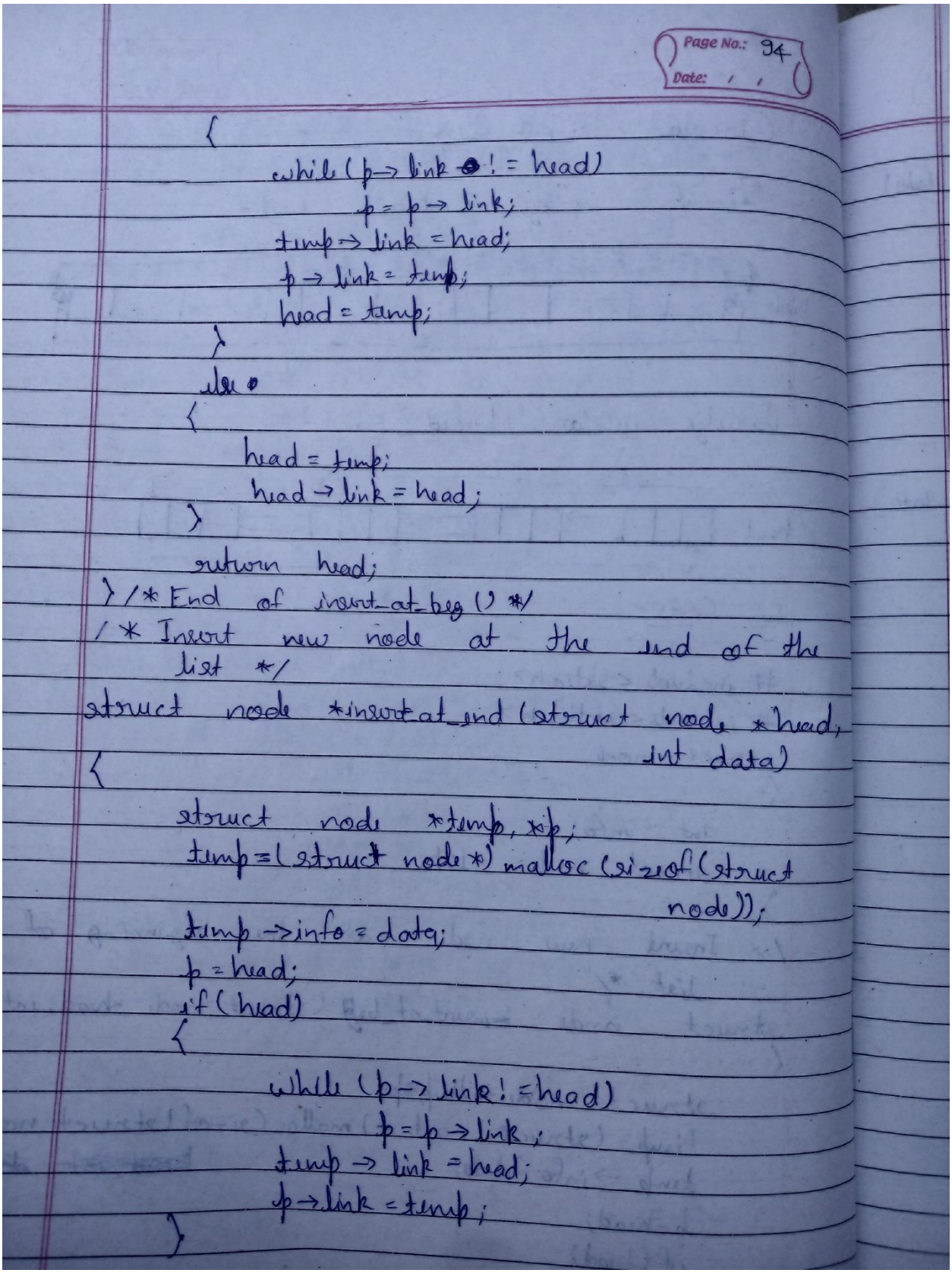


Doubly Circular Linked List -

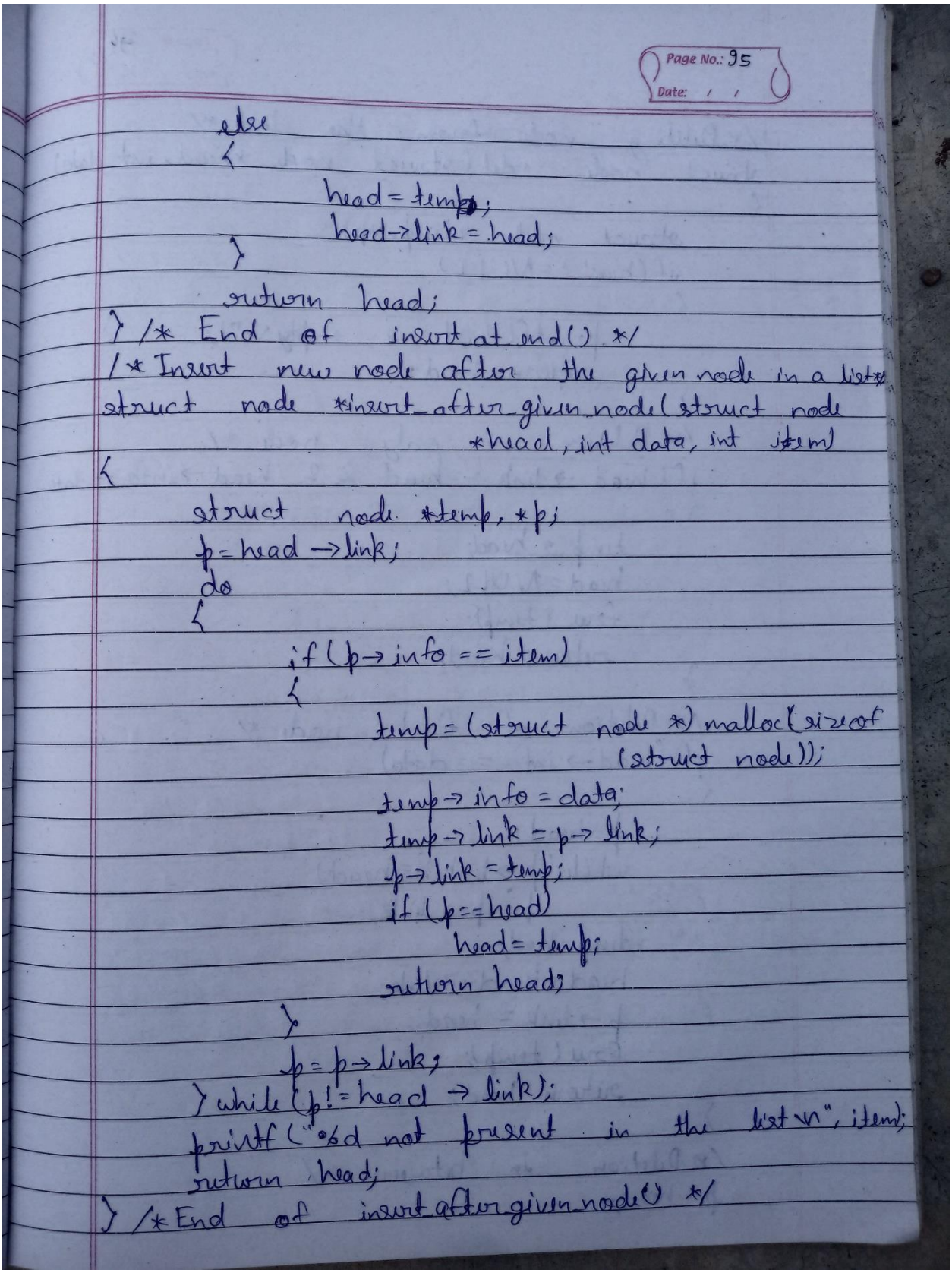


C Code -

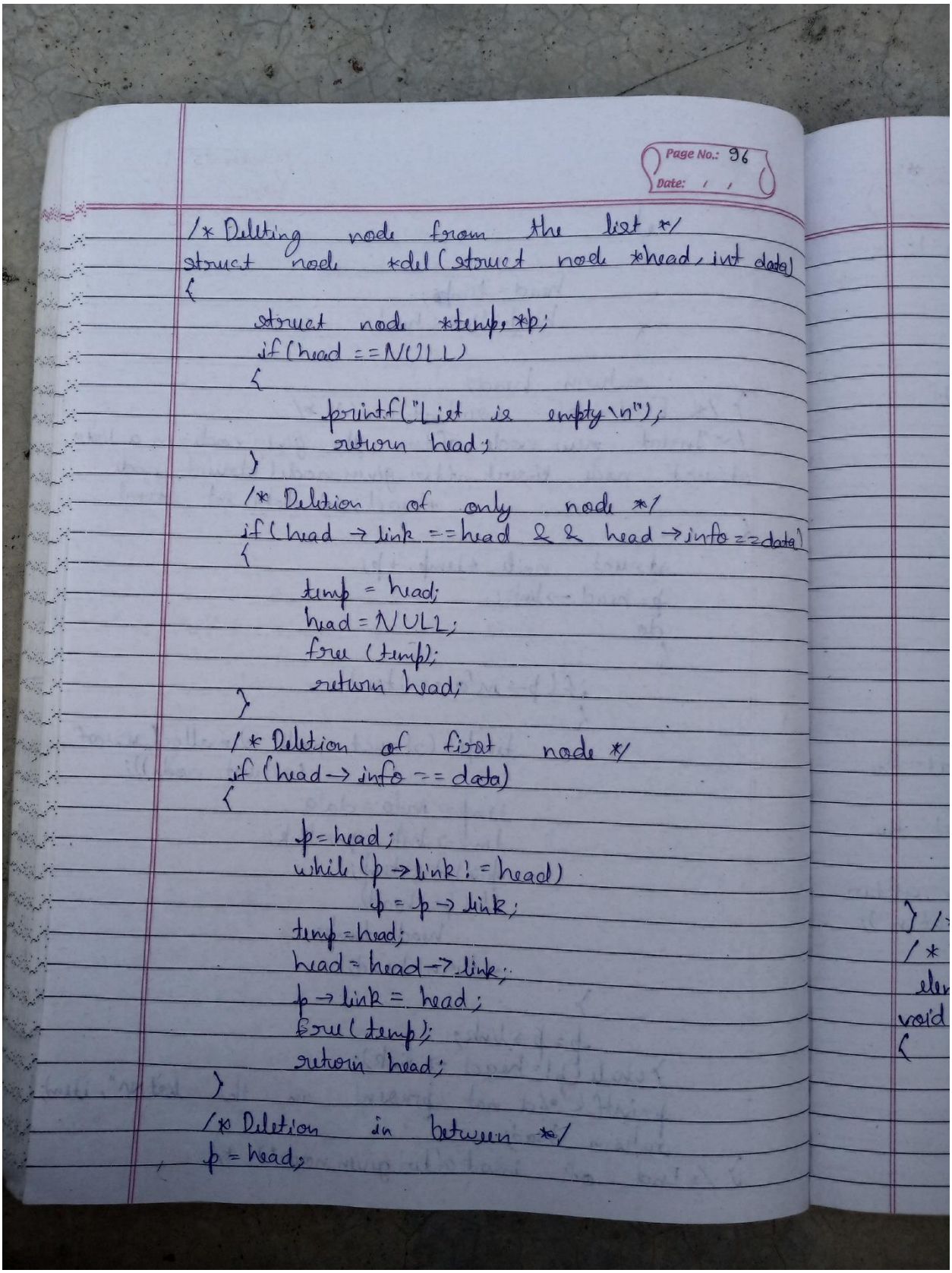
```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int info;
    struct node *link;
};
/* Insert new node at the beginning of the
   list */
struct node *insert_at_beg (struct node *head, int data)
{
    struct node *temp,*p;
    temp = (struct node *) malloc (sizeof (struct node));
    temp -> info = data;
    p=head;
    if (head)
```



```
        {
            while (p->link != head)
                p = p->link;
            temp->link = head;
            p->link = temp;
            head = temp;
        }
        else
        {
            head = temp;
            head->link = head;
        }
        return head;
}/* End of insert_at_beg() */
/* Insert new node at the end of the
   list */
struct node *insert_at_end(struct node *head,
                           int data)
{
    struct node *temp, *p;
    temp=(struct node *) malloc(sizeof(struct
                                node));
    temp->info = data;
    p = head;
    if(head)
    {
        while (p->link!=head)
            p = p->link;
        temp->link = head;
        p->link = temp;
    }
```

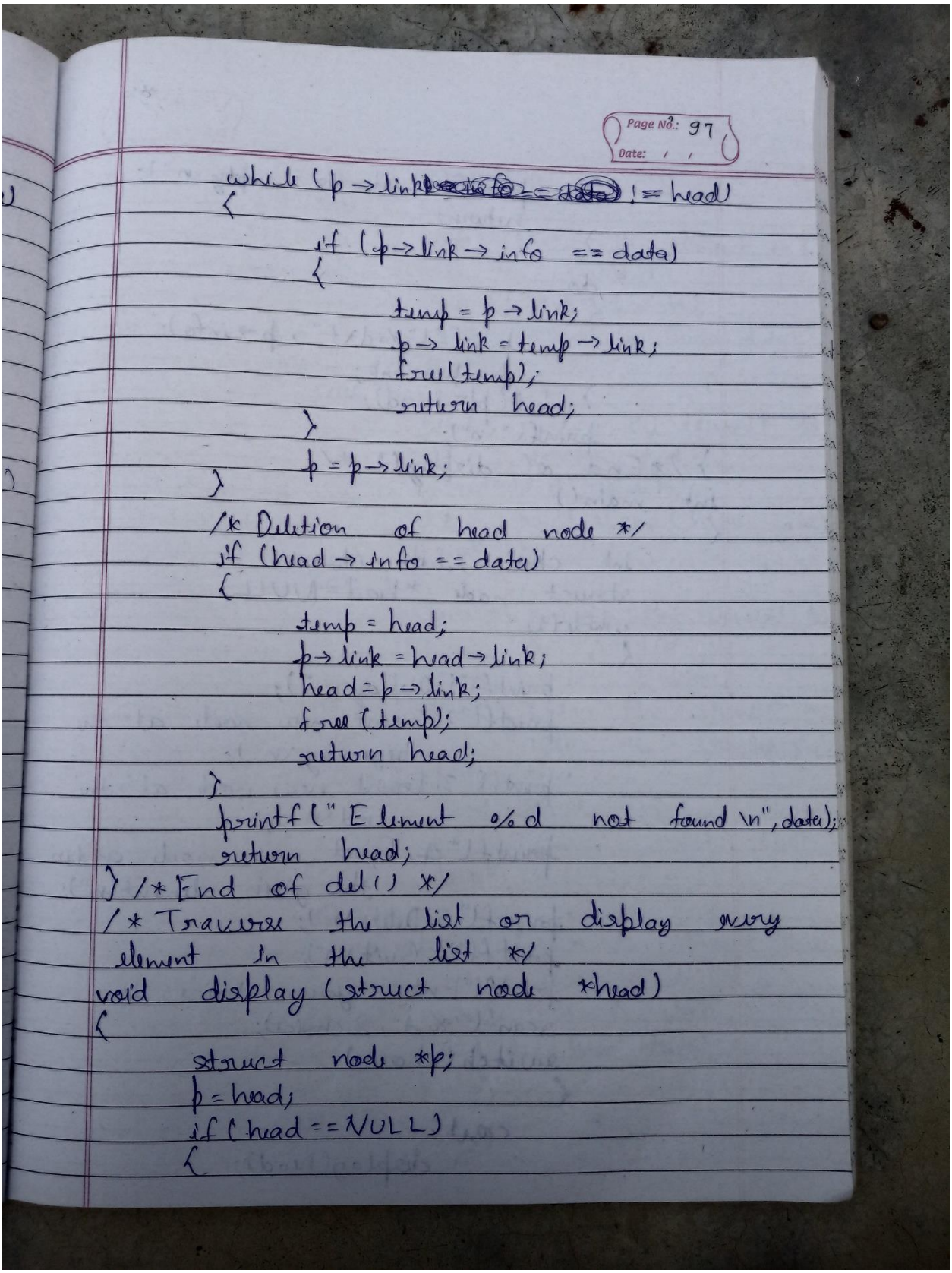


```
Page No.: 95
Date: / /

else
{
        head = temp;
        head->link = head;
}
return head;
} /* End of insert_at_end() */
/* Insert new node after the given node in a list */
struct node *insert_after_given_node(struct node
                    *head, int data, int item)
{
    struct node *temp, *p;
    p = head->link;
    do
    {
        if (p->info == item)
        {
            temp = (struct node *) malloc(sizeof
                        (struct node));
            temp->info = data;
            temp->link = p->link;
            p->link = temp;
            if (p==head)
                head = temp;
            return head;
        }
        p = p->link;
    } while (p != head->link);
    printf("%d not present in the list\n", item);
    return head;
} /* End of insert_after_given_node() */
```



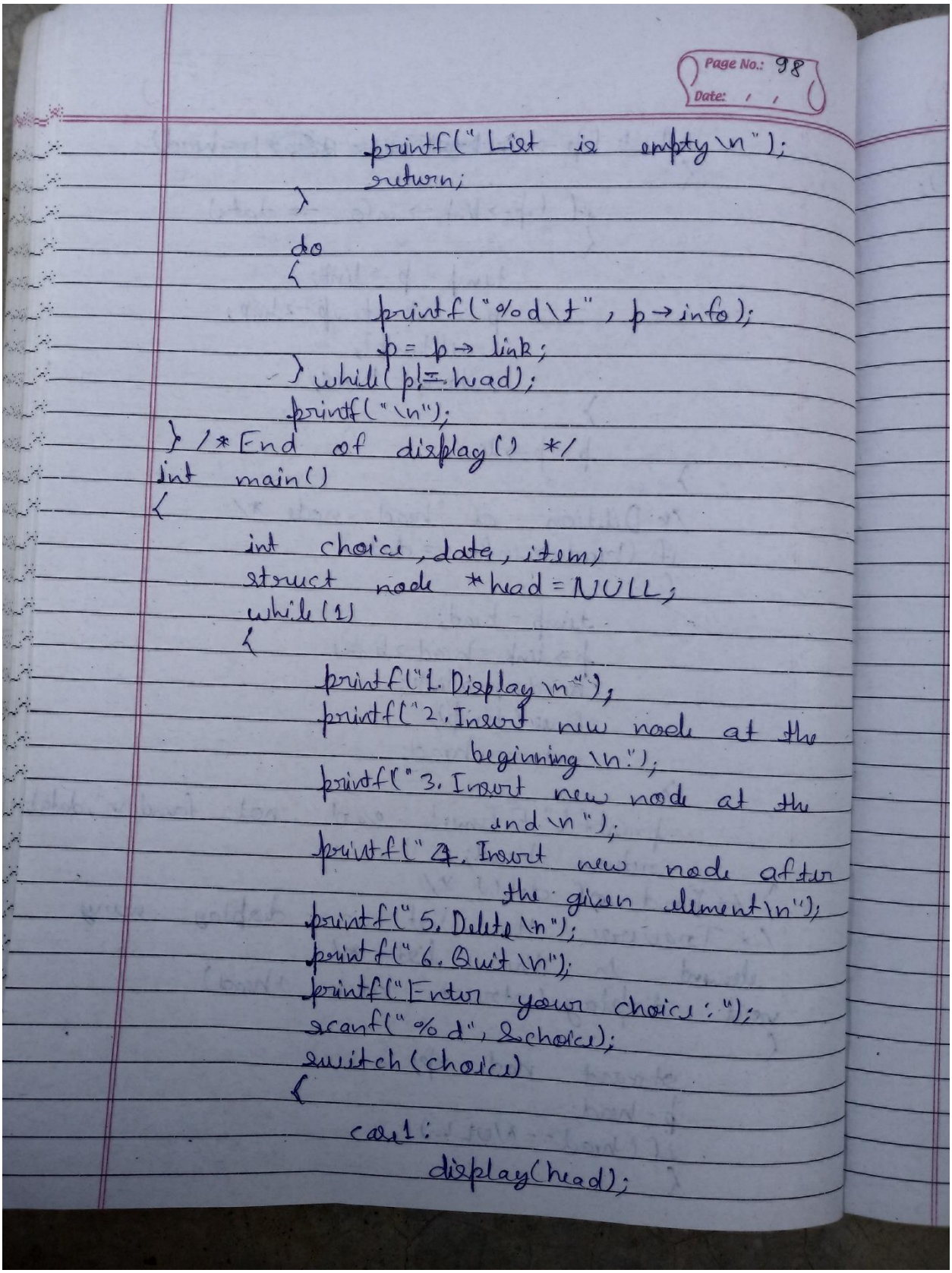
Page No.: 96
Date: / /

```
/* Deleting node from the list */
struct node *del(struct node *head, int data)
{
    struct node *temp, *p;
    if(head == NULL)
    {
        printf("List is empty\n");
        return head;
    }
    /* Deletion of only node */
    if(head->link == head && head->info == data)
    {
        temp = head;
        head = NULL;
        free(temp);
        return head;
    }
    /* Deletion of first node */
    if(head->info == data)
    {
        p = head;
        while(p->link != head)
            p = p->link;
        temp = head;
        head = head->link;
        p->link = head;
        free(temp);
        return head;
    }
    /* Deletion in between */
    p = head;
```

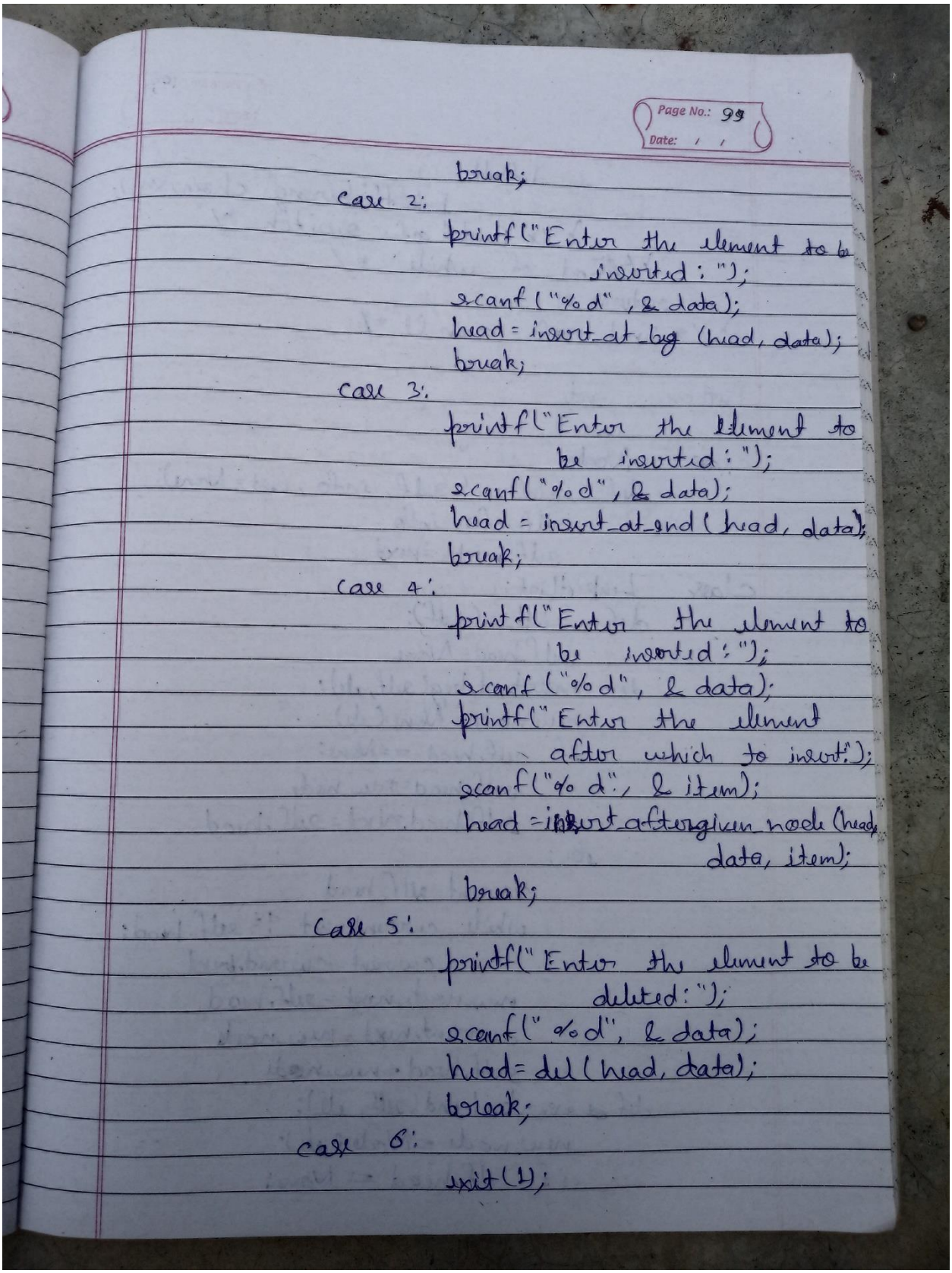


Page No.: 97
Date: / /

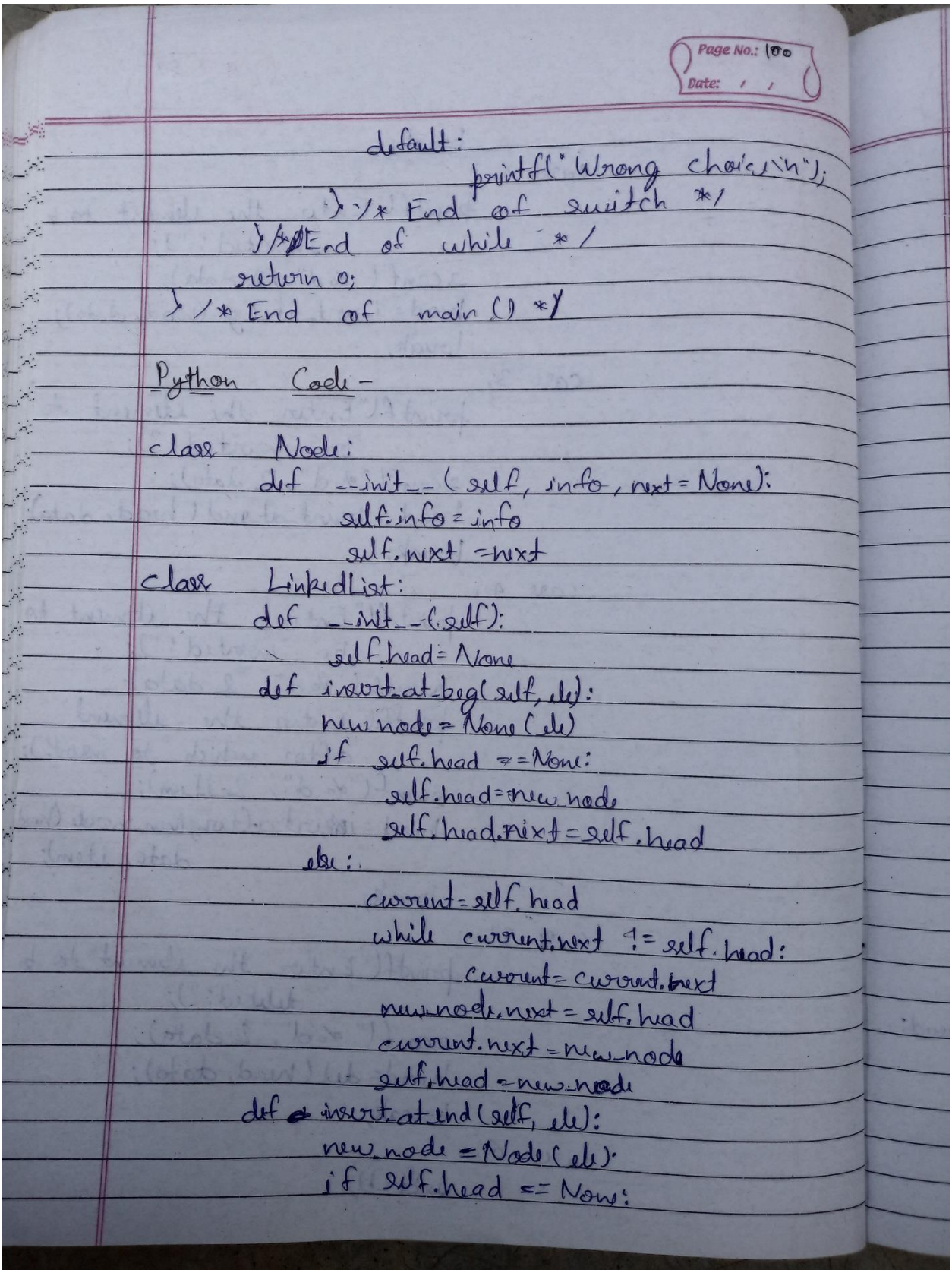
```
while (p->link->info != head)
{
    if (p->link->info == data)
    {
        temp = p->link;
        p->link = temp->link;
        free(temp);
        return head;
    }
    p = p->link;
}
/* Deletion of head node */
if (head->info == data)
{
    temp = head;
    p->link = head->link;
    head = p->link;
    free(temp);
    return head;
}
printf("Element %d not found\n", data);
return head;
} /* End of del() */
/* Traverse the list or display every
 element in the list */
void display(struct node *head)
{
    struct node *p;
    p = head;
    if (head == NULL)
    {
```



```
            printf("List is empty\n");
            return;
        }
        do
        {
            printf("%d\t", p->info);
            p = p->link;
        } while(p!=head);
        printf("\n");
} /* End of display() */
int main()
{
    int choice, data, item;
    struct node *head = NULL;
    while(1)
    {
        printf("1. Display\n");
        printf("2. Insert new node at the
                beginning\n");
        printf("3. Insert new node at the
                end\n");
        printf("4. Insert new node after
                the given element\n");
        printf("5. Delete\n");
        printf("6. Quit\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                display(head);
```



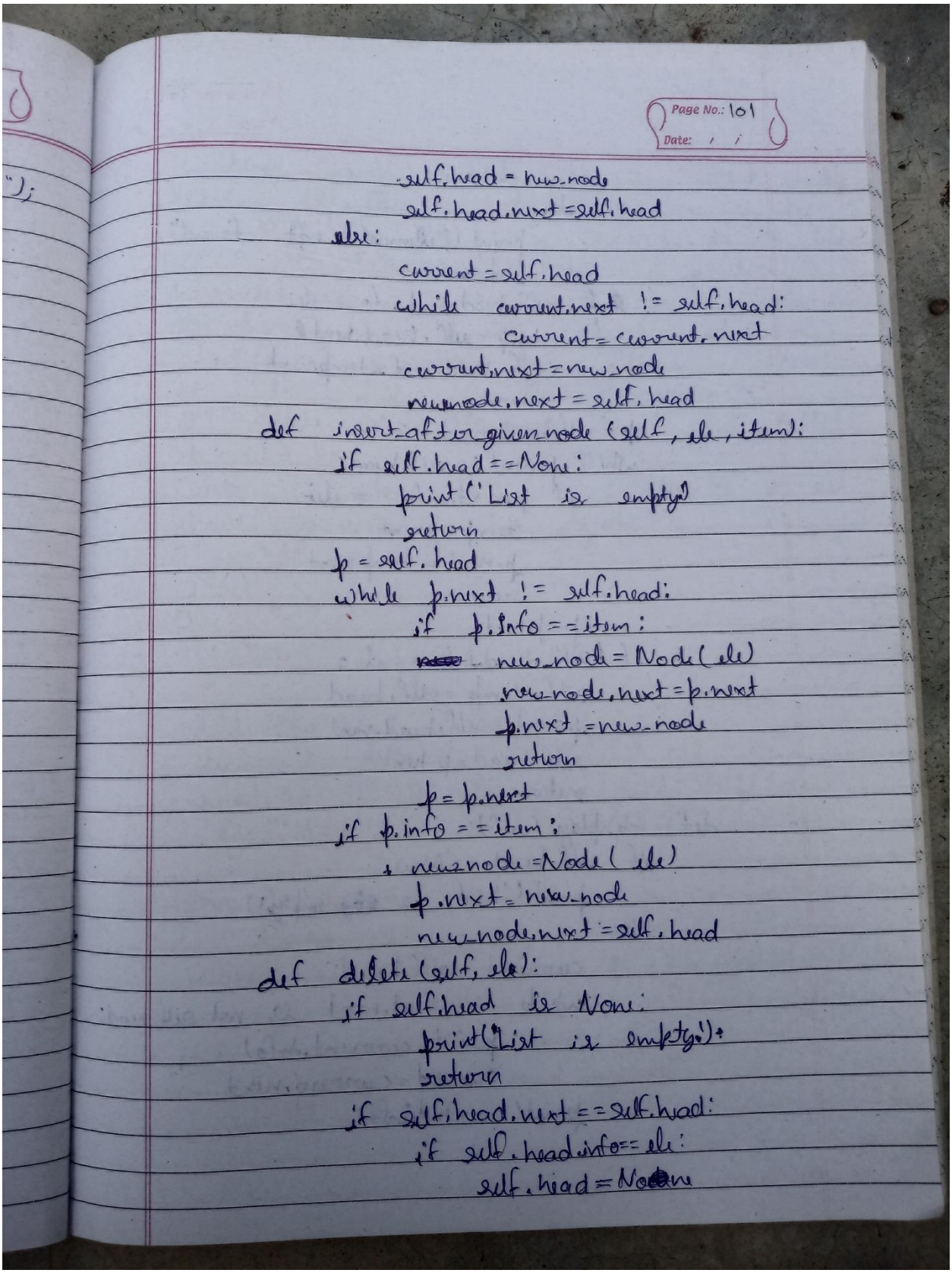
```
Page No.: 99
Date: / /

            break;
        case 2:
            printf("Enter the element to be
                    inserted: ");
            scanf("%d", &data);
            head = insert_at_beg (head, data);
            break;
        case 3:
            printf("Enter the element to
                    be inserted: ");
            scanf("%d", &data);
            head = insert_at_end (head, data);
            break;
        case 4:
            printf("Enter the element to
                    be inserted: ");
            scanf("%d", &data);
            printf("Enter the element
                    after which to insert:");
            scanf("%d", &item);
            head = insert_aftergiven_node (head,
                                    data, item);
            break;
        case 5:
            printf("Enter the element to be
                    deleted: ");
            scanf("%d", &data);
            head = del (head, data);
            break;
        case 6:
            exit(1);
```



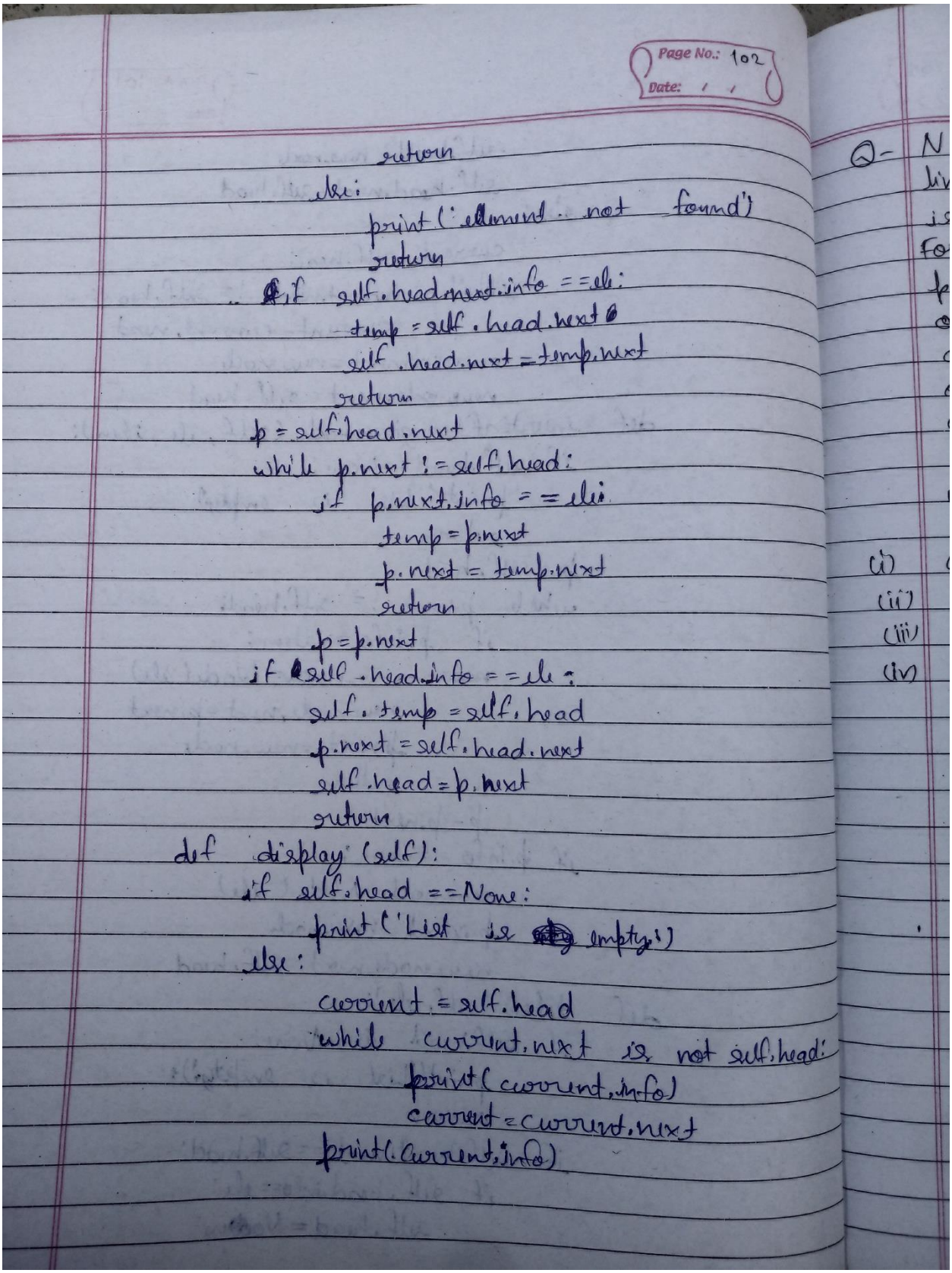
```
        default:
                printf("Wrong choice\n");
        }/* End of switch */
    }/* End of while */
    return 0;
}/* End of main() */
```

Python Code -

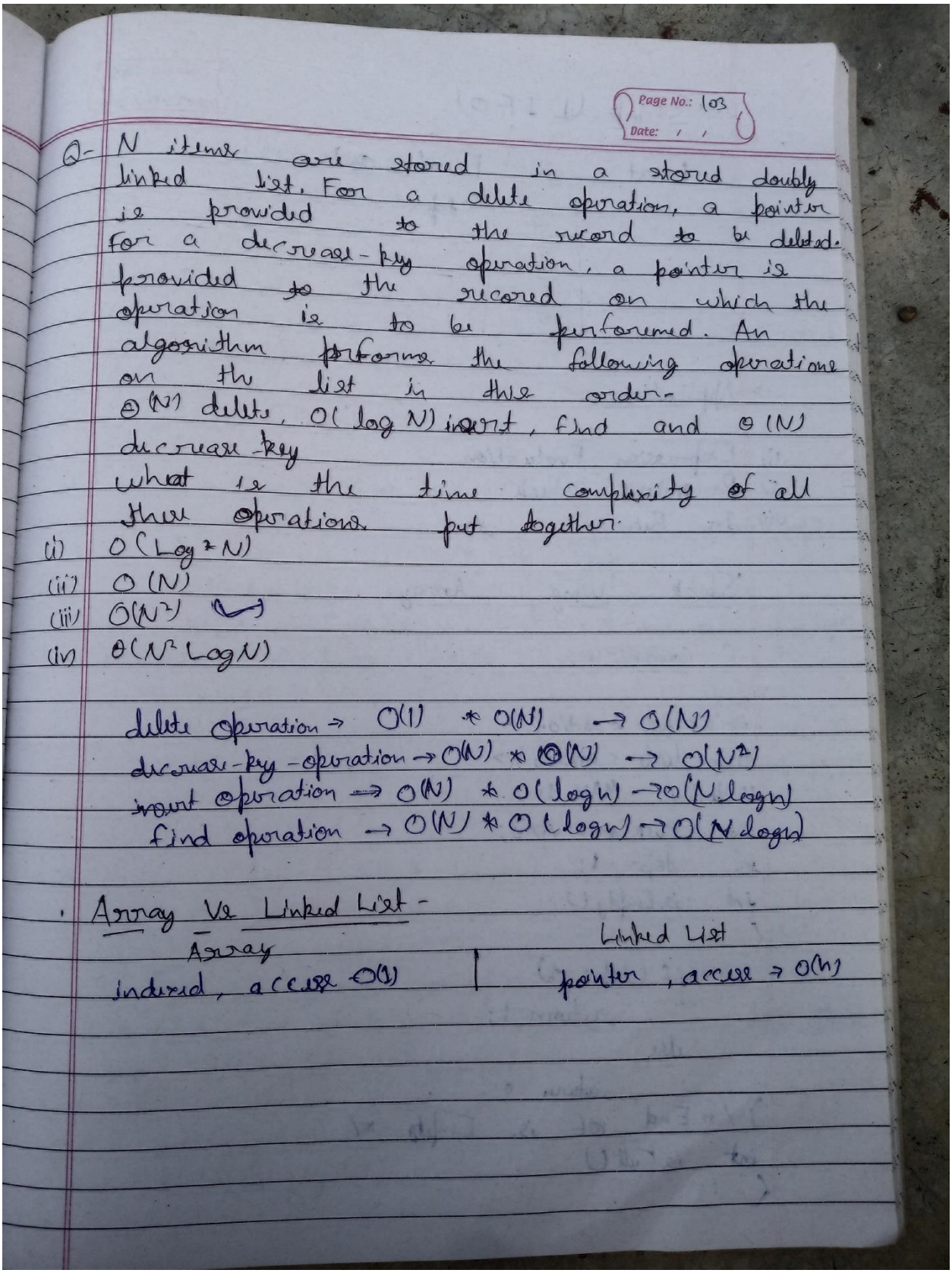
```
class Node:
    def __init__(self, info, next=None):
        self.info = info
        self.next = next
class LinkedList:
    def __init__(self):
        self.head = None
    def insert_at_beg(self, ele):
        new_node = None(ele)
        if self.head == None:
            self.head = new_node
            self.head.next = self.head
        else:
            current = self.head
            while current.next != self.head:
                current = current.next
            new_node.next = self.head
            current.next = new_node
            self.head = new_node
    def insert_at_end(self, ele):
        new_node = Node(ele)
        if self.head == Now:
```



```
            self.head = new_node
            self.head.next = self.head
        else:
            current = self.head
            while current.next != self.head:
                current = current.next
            current.next = new_node
            new_node.next = self.head
    def insert_after_given_node(self, ele, item):
        if self.head == None:
            print('List is empty')
            return
        p = self.head
        while p.next != self.head:
            if p.info == item:
                new_node = Node(ele)
                new_node.next = p.next
                p.next = new_node
                return
            p = p.next
        if p.info == item:
            new_node = Node(ele)
            p.next = new_node
            new_node.next = self.head
    def delete(self, ele):
        if self.head is None:
            print('List is empty.')
            return
        if self.head.next == self.head:
            if self.head.info == ele:
                self.head = None
```



```
                return
            else:
                print('element not found')
                return
        if self.head.next.info == ele:
            temp = self.head.next
            self.head.next = temp.next
            return
        p = self.head.next
        while p.next != self.head:
            if p.next.info == ele:
                temp = p.next
                p.next = temp.next
                return
            p = p.next
        if self.head.info == ele:
            self.temp = self.head
            p.next = self.head.next
            self.head = p.next
            return
    def display(self):
        if self.head == None:
            print('List is empty')
        else:
            current = self.head
            while current.next is not self.head:
                print(current.info)
                current = current.next
            print(current.info)
```



Page No.: 103
Date: / /

Q- N items are stored in a sorted doubly linked list. For a delete operation, a pointer is provided to the record to be deleted. For a decrease-key operation, a pointer is provided to the record on which the operation is to be performed. An algorithm performs the following operations on the list in this order.
$\Theta(N)$ delete, $O(\log N)$ insert, find and $\Theta(N)$ decrease key
what is the time complexity of all these operations put together.

(i) $O(\text{Log}^2 N)$
(ii) $O(N)$
(iii) $O(N^2)$ ✓
(iv) $\Theta(N^2 \text{Log} N)$

delete operation → $O(1) * O(N) \rightarrow O(N)$
decrease-key-operation → $O(N) * \Theta(N) \rightarrow O(N^2)$
insert operation → $O(N) * O(\log N) \rightarrow O(N \log N)$
find operation → $O(N) * O(\log N) \rightarrow O(N \log N)$

- Array Vs Linked List -

Array	Linked List
indexed, access $O(1)$	pointer, access → $O(n)$