09/10/2021

Problem Solving

Applied Prep Course

SATYAM SETH

PART-1

Problem Solving ①

Problems on Arrays

1.1 Find Missing Element ~~to n~~
use ~~n(n+1)/2~~
↳ use XOR
1) First calculate xor of 1 ton
2) Find the XOR of all ~~array~~ array element
3) Return XOR of step① & ② results

1.3 Find Majority Element -

→ use Hash ~~Table~~ Table (Dictionary / Map)
→ use Moore's Voting Algorithm
or ~~Boyer~~ Moore Algo

1) Create 2 variable count=0 and maj
2) traverse the array
~~initialize~~ initialize ~~count~~ maj = ~~0~~ element
if the element is equal to maj
then increment count
else
if count ≠ 0 then count --;
else
count maj = that current array element

1) maj & Count = 0 two variable created.

```
for i=0 to n-1
    if(count==0)
        maj=arr[i]
        count++
    else
        if maj==arr[i]
            c++;
        else
            c--;
```

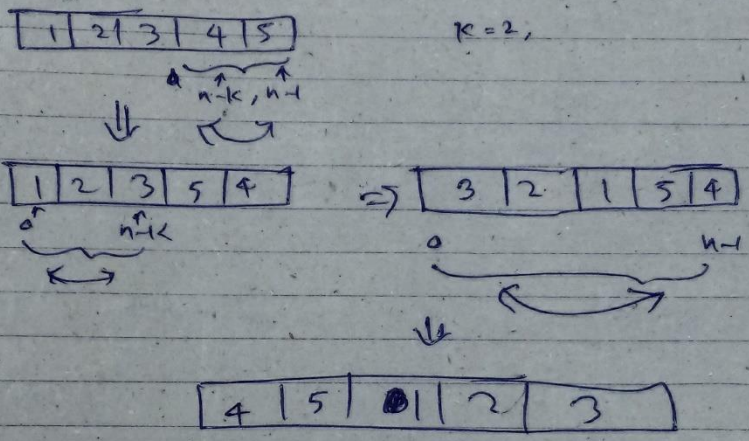
```
if count > n/2
    return maj
else
    return -1
```

(2)

1.4 Rotate Array - k is no of places to rotate.

```
right_Rotate(A, k, n)
{
    reverse(A, n-k, n-1)
    reverse(A, 0, n-k-1)
    reverse(A, 0, n-1)
}
```

Eg

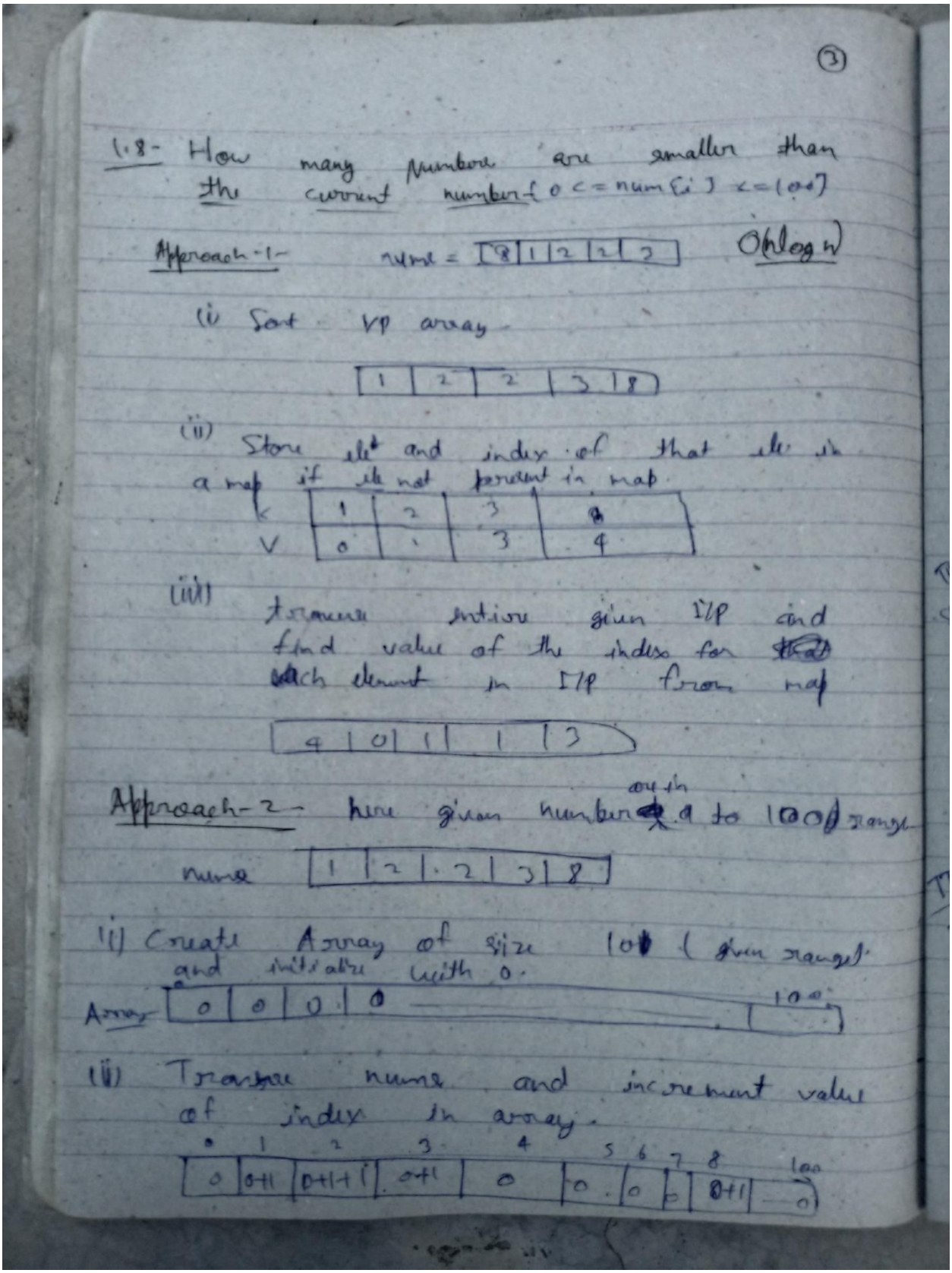


1.5 Single - Number Given a non-empty array of integers, every element appears twice except for one. Find that single one.

Sol 1 - We use Hash Map ~~T(n) = O(n^2)~~ T(n) = O(n), S(n) = O(n)

Sol 2 - Use XOR -

```
ans = 0;
for i = 0 to n-1
    ans = ans XOR a[i]
return ans;
```



③

1.8- How many numbers are smaller than the current number [0 <= nums[i] <= 100]

Approach-1- nums = [8 | 1 | 2 | 2 | 3] O(nlogn)

(i) Sort I/p array

[1 | 2 | 2 | 3 | 8]

(ii) Store ele and index of that ele in a map if ele not present in map.

k	1	2	3	8
v	0	1	3	4

(iii) Traverse entire given I/p and find value of the index for each element in I/p from map

[4 | 0 | 1 | 1 | 3]

Approach-2- here given number out in 0 to 100 range

nums [1 | 2 | 2 | 3 | 8]

(i) Create Array of size 101 (given range) and initialize with 0.

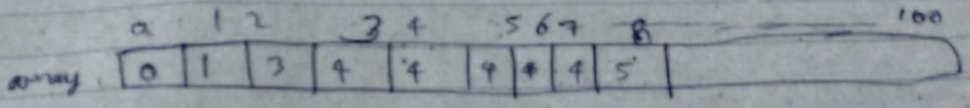
Array [0 | 0 | 0 | 0 ... | 100]

(ii) Traverse nums and increment value of index in array.

0	1	2	3	4	5	6	7	8	100
0	0+1	0+1+1	0+1	0	0	0	0	0+1	0

④

(iii) Perform Prefix Sum-



(iv) return for any element x in nums array[x-1].

T.C = O(n), Space Complexity - O(1)

1.19 Sort Array by Parity-

TC = O(n) SC = O(n) Approach-1 run two loop and check even no. in first loop and push back in an aux array & in the second loop chicke odd numbers and push back in same aux array then return array.

Approach-2- Use 2-pointer

TC: O(n) SC = O(1)

```
int i=0, j=A.size()-1;
while(i<j)
{
    if(A[i]%2!=0)
    {
        swap(A[i], A[j])
        j--;
    }
    else
    i++;
}
return A;
```

Ex- [4|3|2|9]

i j

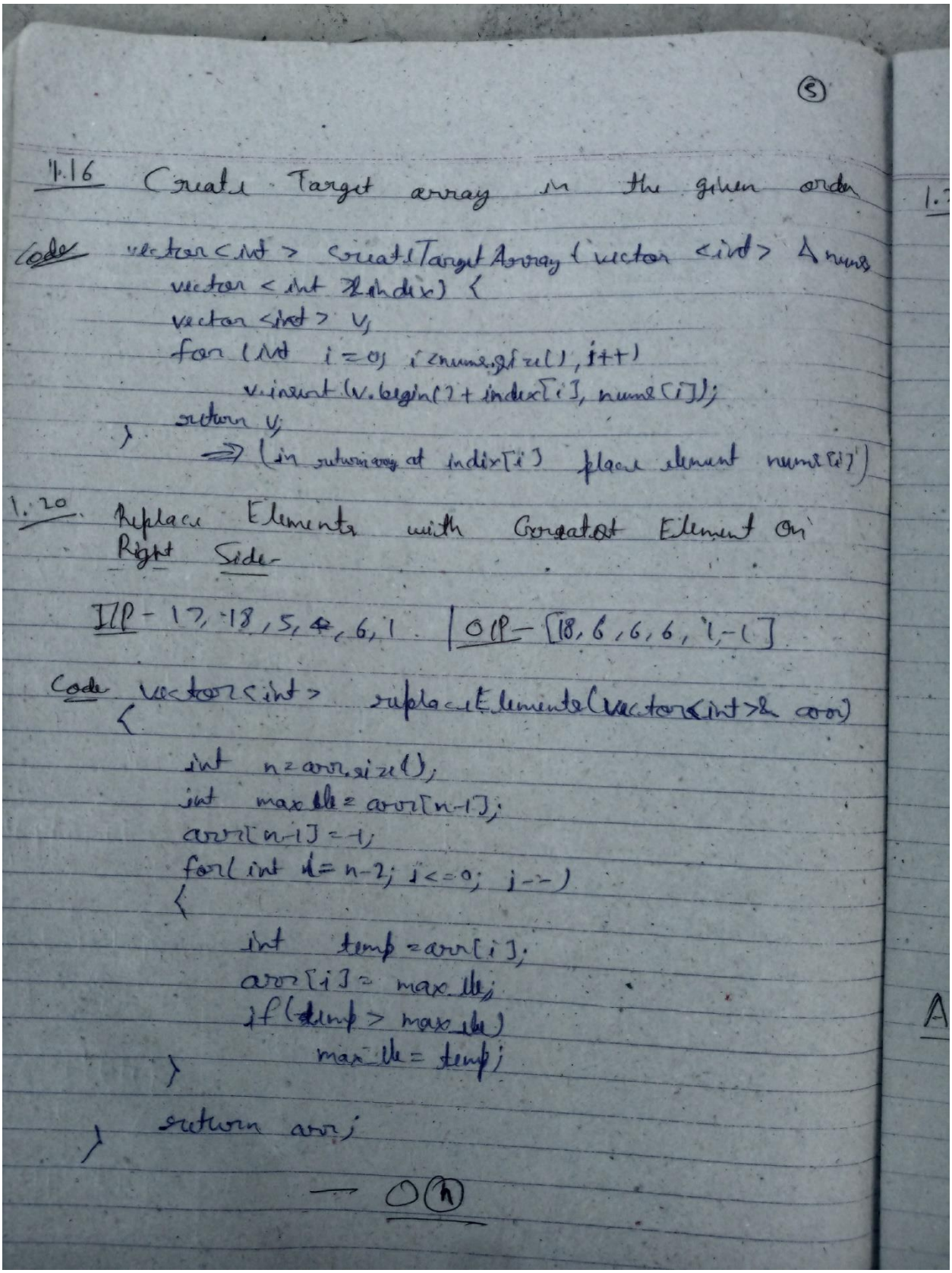
[4|3|2|9]

i j

[4|9|2|3]

i j

[4|2|9|3]



⑤

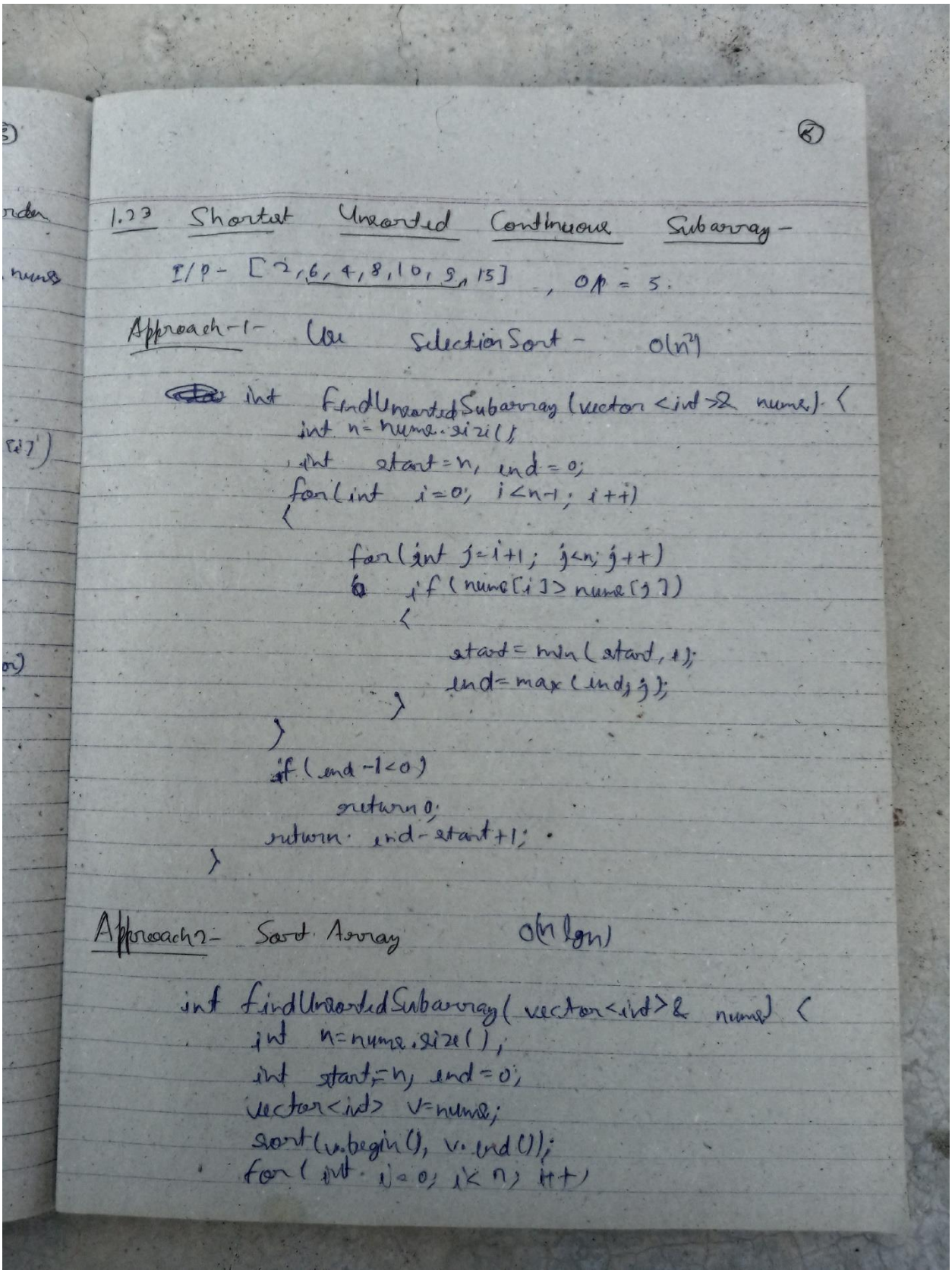
1.16 Create Target array in the given order

Code
```
vector<int> createTargetArray(vector<int>& nums,
vector<int>& index) {
vector<int> v;
for(int i=0; i<nums.size(); i++)
    v.insert(v.begin()+index[i], nums[i]);
return v;
}
```
⇒ (in return array at index[i] place element nums[i])

1.20 Replace Elements with Greatest Element on Right Side

I/P - 17, 18, 5, 4, 6, 1 | O/P - [18, 6, 6, 6, 1, -1]

Code
```
vector<int> replaceElements(vector<int>& arr)
{
    int n=arr.size();
    int max_ele=arr[n-1];
    arr[n-1]=-1;
    for(int i=n-2; i<=0; i--)
    {
        int temp=arr[i];
        arr[i]=max_ele;
        if(temp>max_ele)
            max_ele=temp;
    }
return arr;
}
```
→ O(n)



1.23 Shortest Unsorted Continuous Subarray -

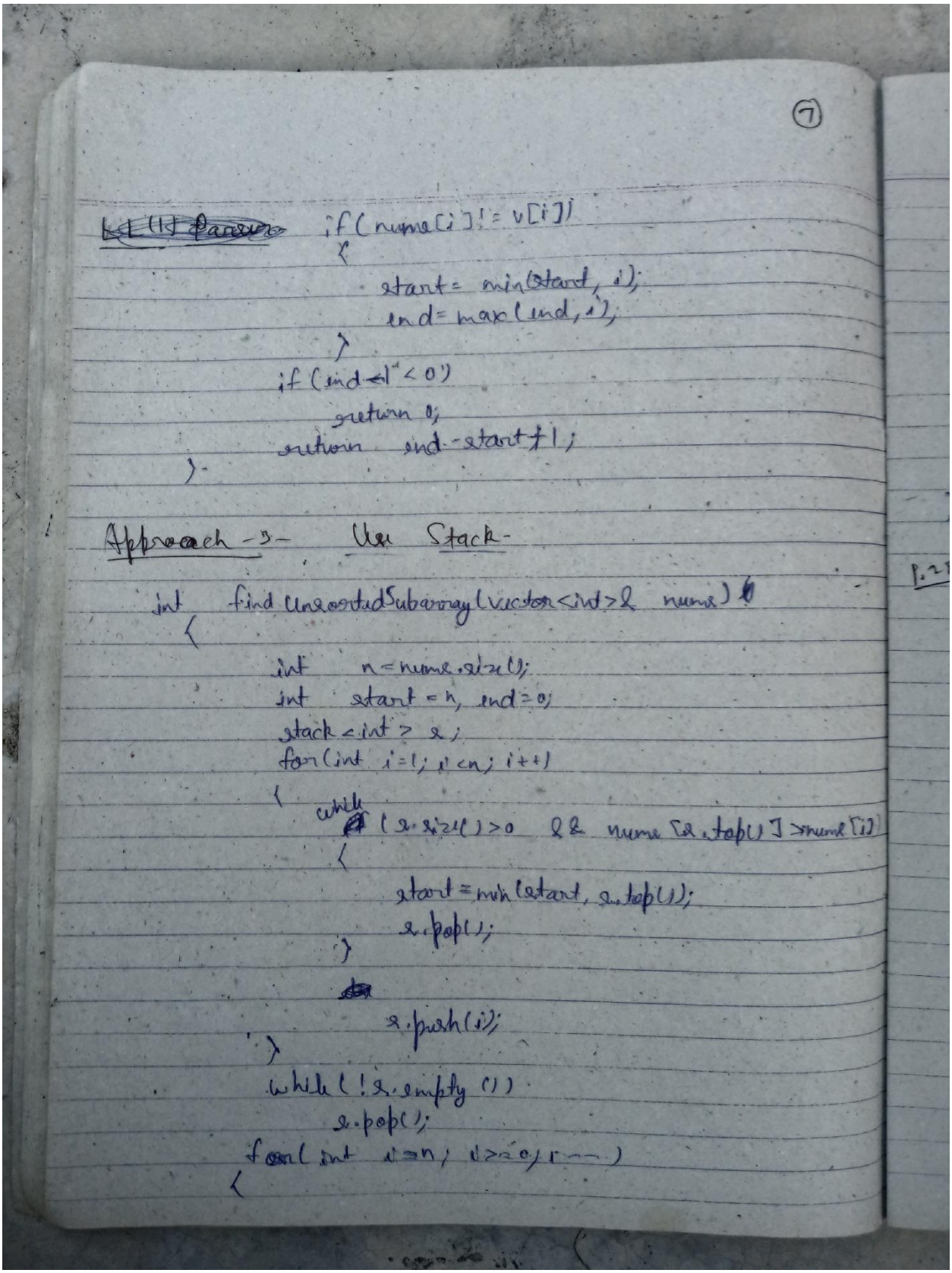
I/P - [2, 6, 4, 8, 10, 9, 15], O/P = 5.

Approach-1- Use Selection Sort - $O(n^2)$

```
int FindUnsortedSubarray(vector<int>& nums) {
    int n = nums.size();
    int start = n, end = 0;
    for(int i=0; i<n-1; i++)
    {
        for(int j=i+1; j<n; j++)
        {
            if(nums[i] > nums[j])
            {
                start = min(start, i);
                end = max(end, j);
            }
        }
    }
    if(end-1<0)
        return 0;
    return end-start+1;
}
```

Approach-2- Sort Array $O(n \log n)$

```
int findUnsortedSubarray(vector<int>& nums) {
    int n=nums.size();
    int start=n, end=0;
    vector<int> v=nums;
    sort(v.begin(), v.end());
    for(int i=0; i<n; i++)
```



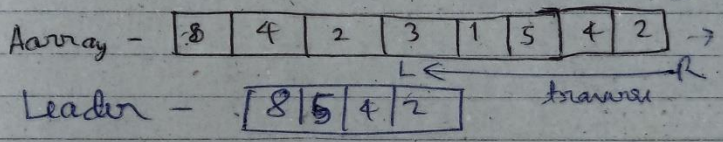
```
if(nums[i] != v[i])
{
    start = min(start, i);
    end = max(end, i);
}
if(end-1 < 0)
    return 0;
return end-start+1;
}
```

Approach - 3 - Use Stack -

```
int findUnsortedSubarray(vector<int>& nums) {
{
    int n = nums.size();
    int start = n, end = 0;
    stack<int> s;
    for(int i=1; i<n; i++)
    {
        while(s.size()>0 && nums[s.top()] > nums[i])
        {
            start = min(start, s.top());
            s.pop();
        }
        s.push(i);
    }
    while(!s.empty())
        s.pop();
    for(int i=n; i>=0; i--)
    {
```

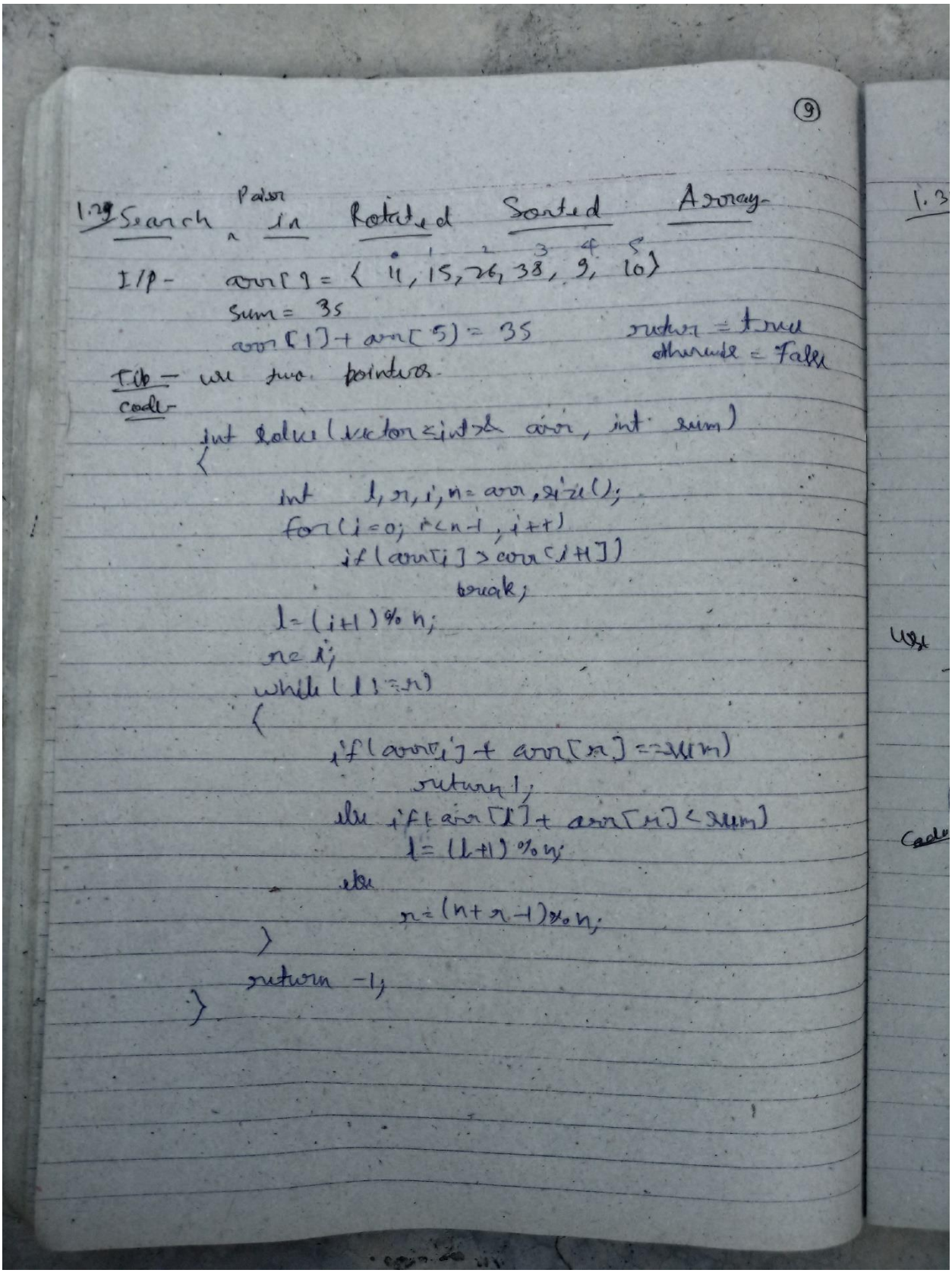
```
while( !s.empty() && nums[s.top()]<nums[i])
{
    end=max (end, s.top());
    s.pop();
}
s.push(i);
}
if (end - start >= 0)
    return end - start +1;
return 0;
}
```

P.28 Find Leaders in an array- O(n)



Pseudo code-

```
find Leaders (A, n)
    current_Max = -∞
    for i = n-1 to 0
        if a[i] > currentMax
            print A[i]
            currentMax = A[i]
```



⑨

1.29 Search a Pair in Rotated Sorted Array-

I/P - arr[] = { 11, 15, 26, 38, 9, 10 }
(1) (2) (3) (4) (5)
Sum = 35
arr[1] + arr[5] = 35 return = true
otherwise = False

T.C - we two pointers.

Code-

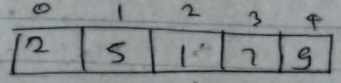
```
int Solve(vector<int>& arr, int sum)
{
    int l, r, i, n = arr.size();
    for(i=0; i<n-1; i++)
        if(arr[i] > arr[i+1])
            break;
    l = (i+1) % n;
    r = i;
    while(l != r)
    {
        if(arr[l] + arr[r] == sum)
            return 1;
        else if(arr[l] + arr[r] < sum)
            l = (l+1) % n;
        else
            r = (n+r-1) % n;
    }
    return -1;
}
```

(10)

1.31 Inversion in an array

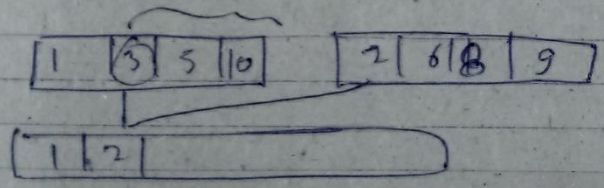
Inversion- $i<j$ and $A[i] > A[j]$

Tip- If array is sorted then there is no any inversion



$i=0$, $j=2$	$i=1$, $j=2$
$2>1$ (1)	$5>1$ (2)

Use Merge in Merge Sort



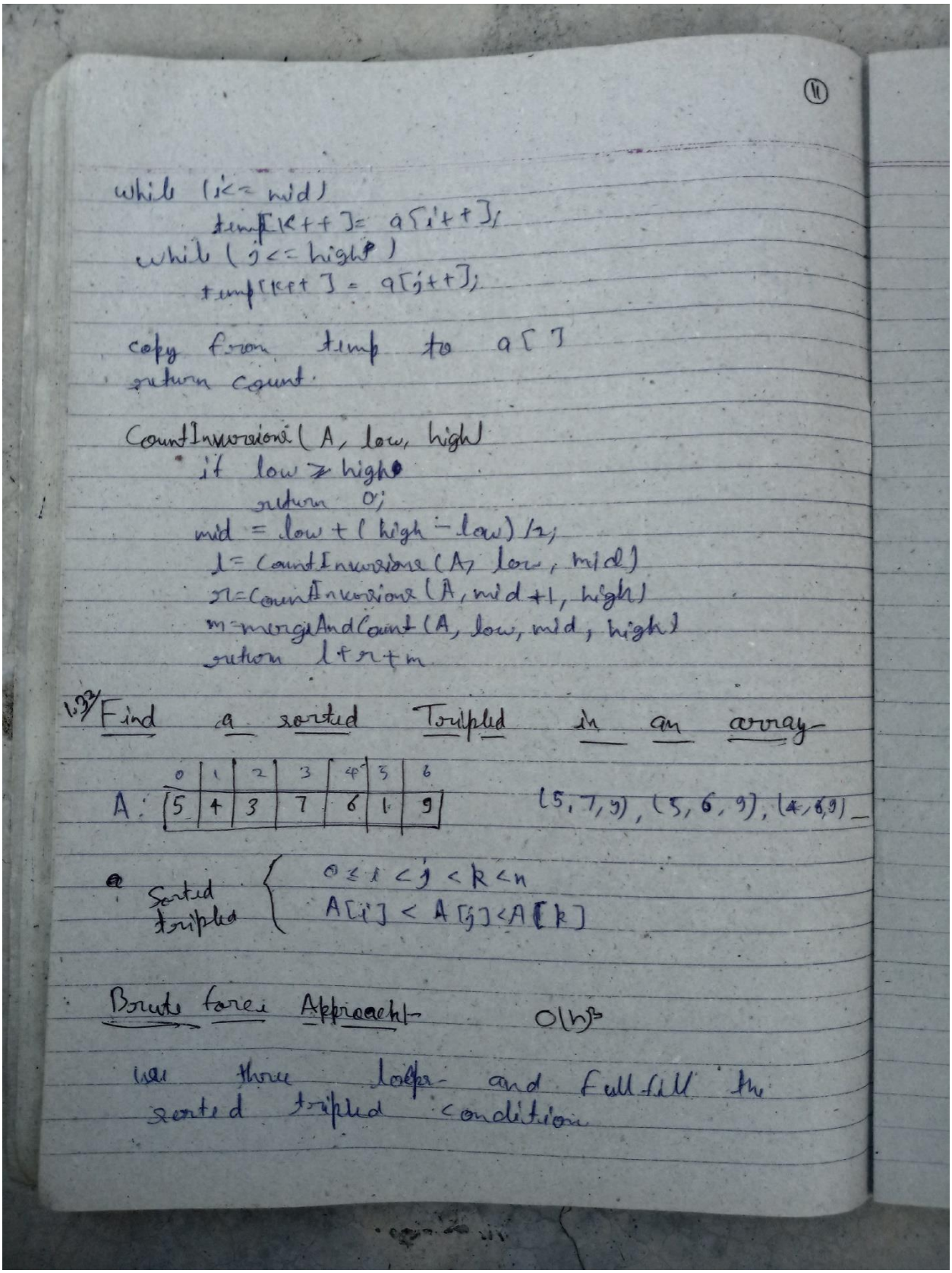
$i<j$, $A[i] > A[j]$

Count = Count + (mid - i + 1)

Code-

MergeAndCount(A, low, mid, high) - $T(n) = O(n \log n)$, $S(n) = O(n \log n)$

```
count = 0; i = low, j = mid+1; k = 0
Copy A[low ---- high] into temp[]
while (i<=mid && j<= high)
    if a[i] > a[j]
        count += (mid - i + 1)
        temp[k++] = a[i++]
    else
        temp[k++] = a[j++]
```



```
while (i <= mid)
    temp[k++] = a[i++];
while (j <= high)
    temp[k++] = a[j++];

copy from temp to a[]
return count.

CountInversions(A, low, high)
    if low >= high
        return 0;
    mid = low + (high - low)/2;
    l = CountInversions(A, low, mid)
    r = CountInversions(A, mid+1, high)
    m = mergeAndCount(A, low, mid, high)
    return l+r+m
```

1.23/ Find a sorted Tripled in an array

	0	1	2	3	4	5	6
A:	5	4	3	7	6	1	9

(5, 7, 9), (5, 6, 9), (4, 6, 9)

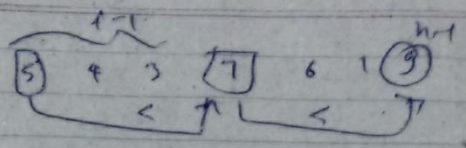
a. Sorted tripled $\begin{cases} 0 \le i < j < k < n \\ A[i] < A[j] < A[k] \end{cases}$

Brute force Approach - $O(n^3)$

use three loops and fulfill the sorted tripled condition

(12)

Approach-2 $O(n^2)$




```
for i=0 to n-1
    for j = 0 to i-1
        find if there exists a num < a[i]
    for j=i+1 to n-1
        find an element > a[i]
```

Approach - 3 $O(n)$

```
FindTriplet(A, n)
    if n<3 return NULL
    min_index = 0
    low=mid = -1
    for i=1 to n-1
        if a[i] < a[min_index]
            min_index = i
        else if mid=-1
            low= min_index
            mid = i
        else if a[i] < a[mid]
            low = min_index
            mid = i
        else
            return (low, mid, i)
```

(13)

1.33 Partition Equal Sum - See Aditya verma Equal Subset Sum Partition

1.34 Array Product Problem - Solve without use Division

I/P - [1, 2, 3, 4] ⇒ O/P - [24, 12, 8, 6]

(2×3×4) (1×3×4) (1×2×4) (1×2×3)

return array o/p such that o/p[i] is equal to the product of all the elements of nums except nums[i].

Constraint - all nums[i] > 1

Approach - ans[.] O(n²)

```
for i = 0 to n-1
    L = 1
    for j = 0 to i-1
        L = L * Arr[j]
    R = 1
    for j = i+1 to n-1
        R = R * Arr[j]
    ans[i] = L * R
```

Approach-2 Tm = O(n) S(n) = O(n)

i=0 → n-1

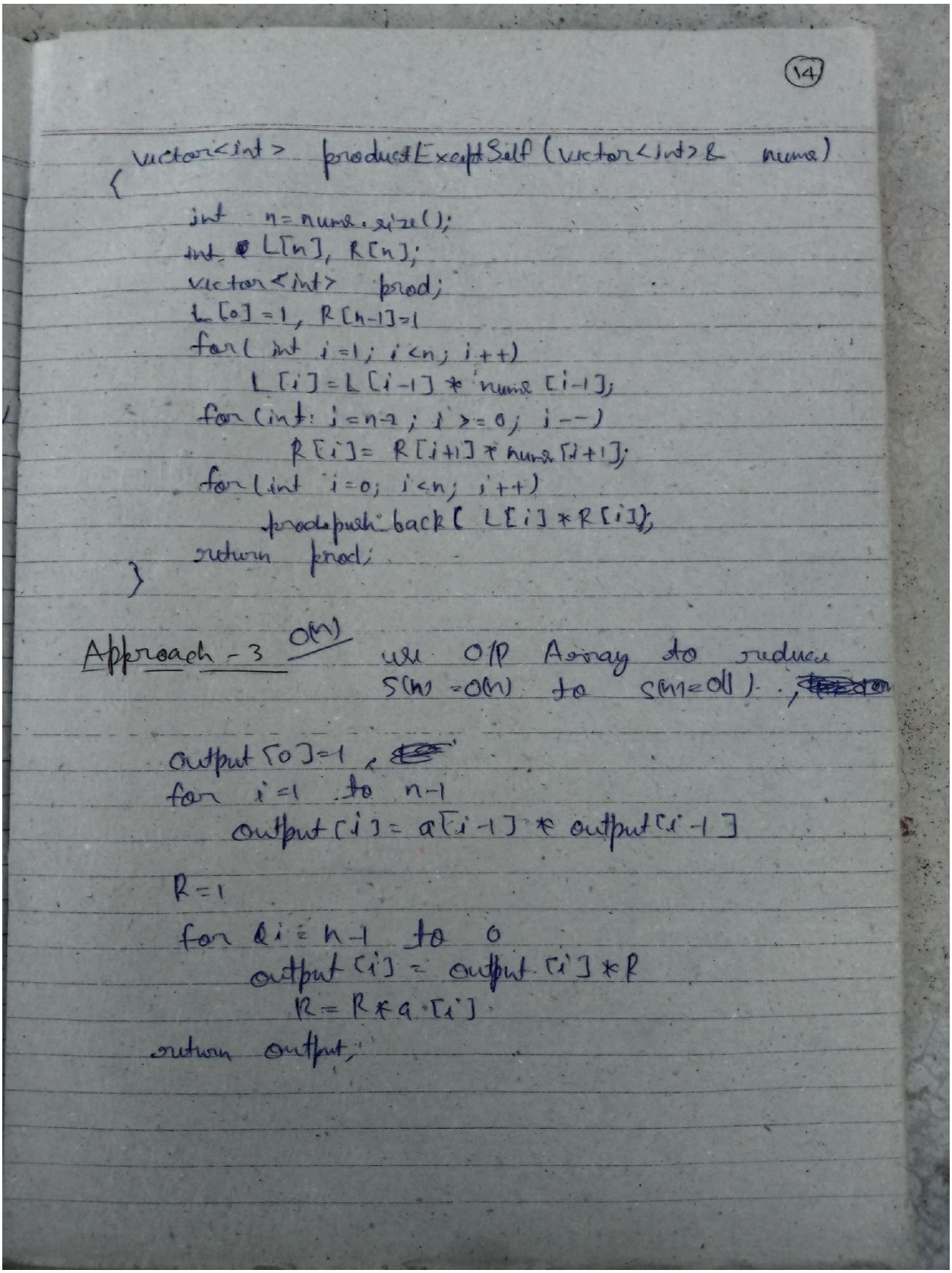
A: [4, 5, 1, 8, 2, 10, 6]

0 ← n-1

L: {1, 4, 20, 20, 160, 320, 320}

R: {480, 96, 96, 12, 6, 6, 1}

O - L[.] × R[i]



(14)

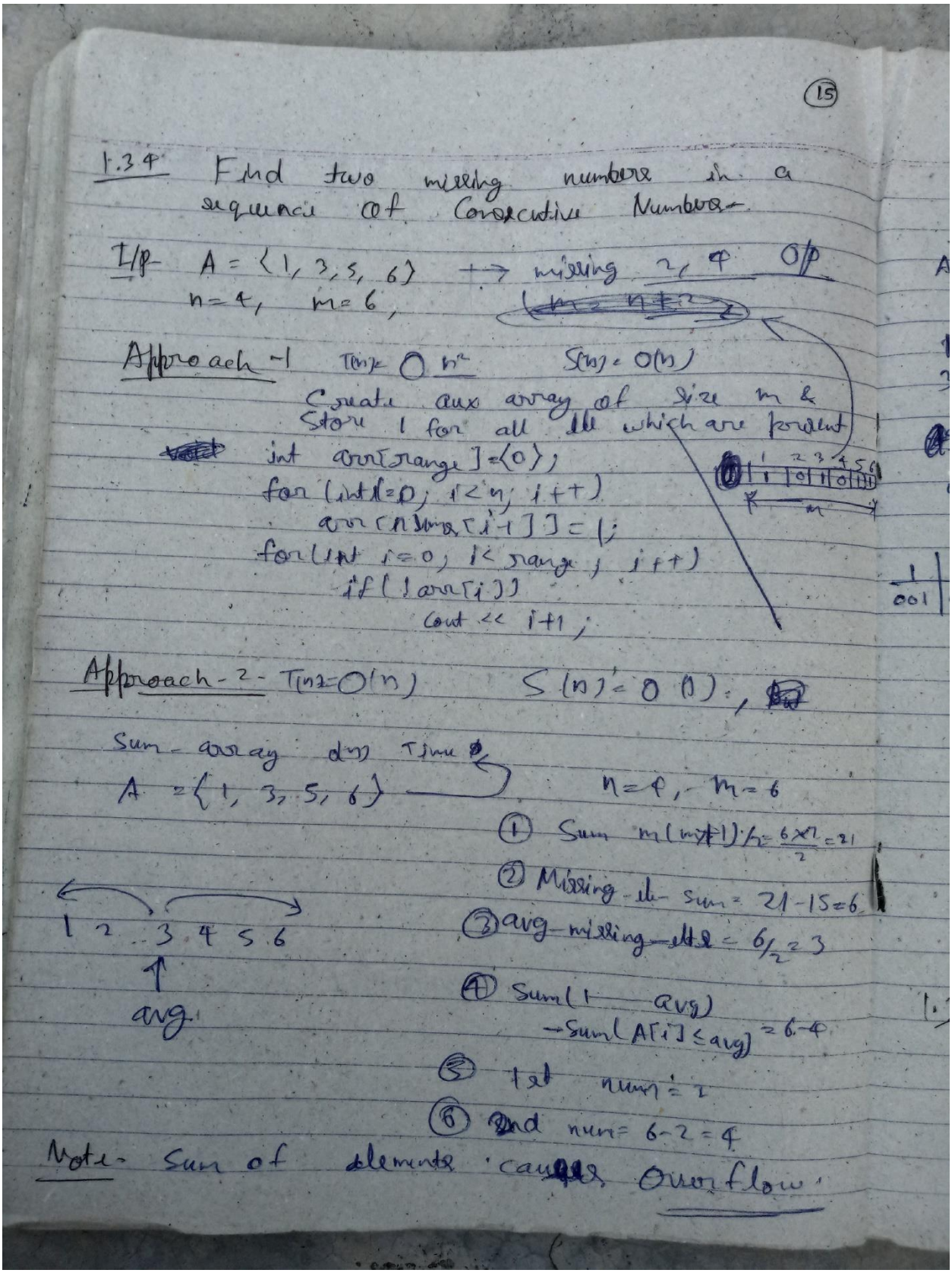
```
vector<int> productExceptSelf(vector<int>& nums)
{
    int n = nums.size();
    int L[n], R[n];
    vector<int> prod;
    L[0] = 1, R[n-1] = 1
    for( int i = 1; i < n; i++)
        L[i] = L[i-1] * nums[i-1];
    for(int i = n-2; i >= 0; i--)
        R[i] = R[i+1] * nums[i+1];
    for(int i = 0; i < n; i++)
        prod.push_back( L[i] * R[i]);
    return prod;
}
```

Approach - 3 O(n)

use O/P Array to reduce S(n) = O(n) to S(n) = O(1).

```
output[0] = 1
for i = 1 to n-1
    output[i] = a[i-1] * output[i-1]

R = 1
for i = n-1 to 0
    output[i] = output[i] * R
    R = R * a[i]
return output;
```



(15)

1.34 Find two missing numbers in a sequence of Consecutive Numbers.

I/p A = {1, 3, 5, 6} → missing 2, 4 O/p

n = 4, m = 6,

Approach -1 Time O(n²) S(n) = O(n)

Create aux array of size m & store 1 for all the which are present

```
int arr[range] = {0};
for (int i=0; i<n; i++)
    arr[nums[i]] = 1;
for (int i=0; i<range; i++)
    if (!arr[i])
        cout << i+1;
```

Approach-2 - T(n) = O(n) S(n) = O(1),

Sum - array dry Time

A = {1, 3, 5, 6} n = 4, m = 6

① Sum m(m+1)/2 = 6×7/2 = 21

② Missing el - sum = 21 - 15 = 6

③ avg - missing el = 6/2 = 3

④ Sum(1 ... avg) - Sum(A[i] ≤ avg) = 6 - 4

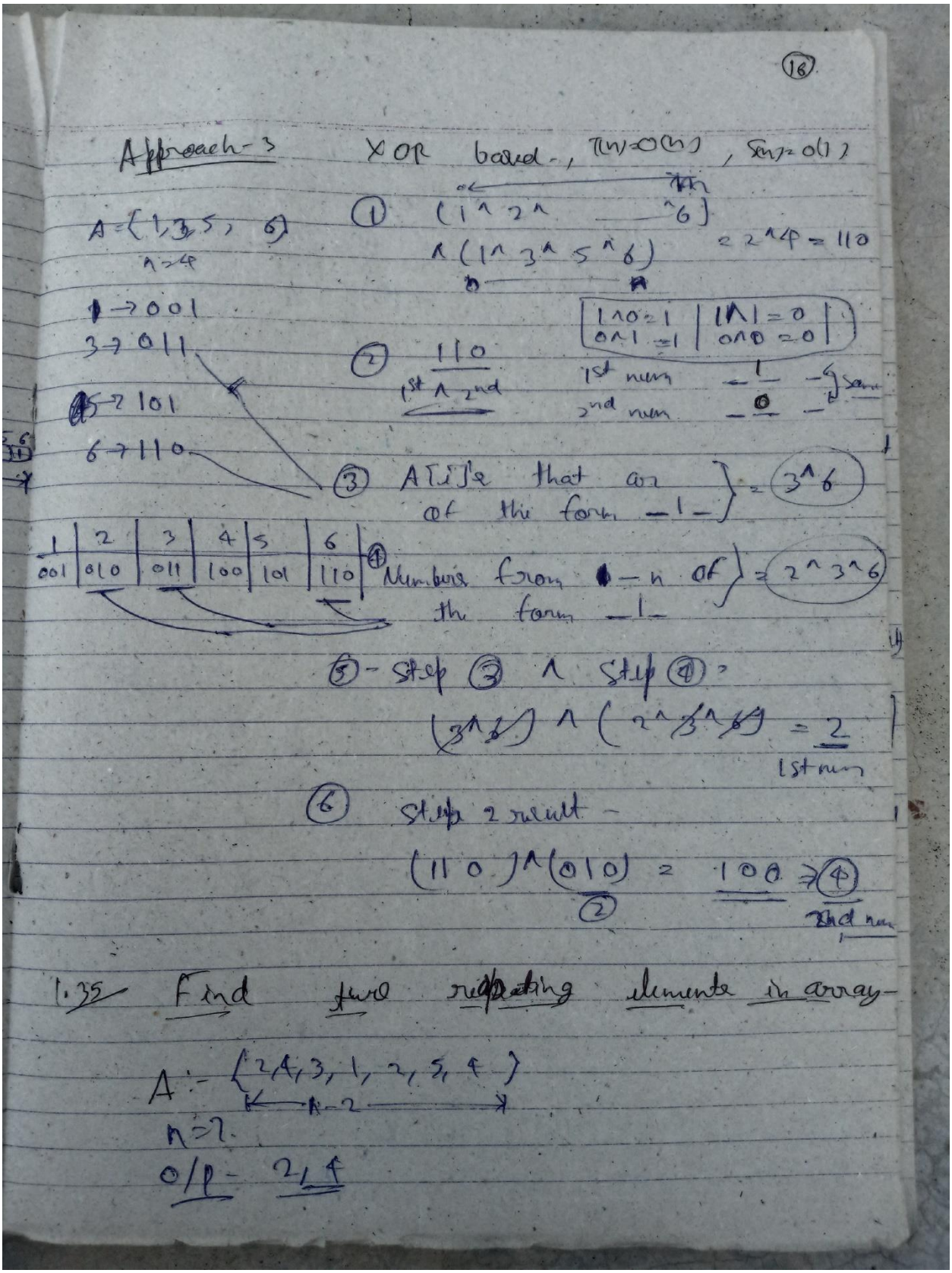
⑤ 1st num = 1

⑥ 2nd num = 6 - 2 = 4

1 2 3 4 5 6

↑ avg

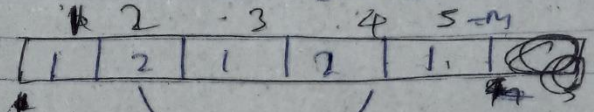
Note - Sum of elements can cause Overflow.


16
Approach-3 XOR based -, T(n)=O(n), S(n)=O(1)
A={1,3,5,6}
n=4
1→001
3→011
5→101
6→110
1 2 3 4 5 6
001 010 011 100 101 110
1st ^ 2nd
1st num
2nd num
Same
Numbers from 1→n of the form _1_
5- Step 3 ^ Step 4:
1st num
Step 2 result -
2nd num
1.35 Find two repeating elements in array-
A:- {2,4,3,1,2,5,4}
n=7.
O/P- 2,4

(17)

Approach 1- Use aux array of size (n-2) to store no. of occurence of each element & return elements for that no. of occurrence is 2.

$m = n-2 = 5$



O/P <- 2, 4

T(n) = O(n)
S(n) = O(n)

Approach 2- O(n) time, O(1) space.

A = {2, 4, 3, 1, 2, 5, 4}

n = 7 , m = n-2 = 5

Let x, y are required elements

① $S' = \frac{m(m+1)}{2} = 15$

$P' = m! = 5!$

overflow int

② $x+y = S - S' = 6$

$x \cdot y = P/P' = 8$

③ $(x-y)^2 = (x+y)^2 - 4xy$

$x-y = \sqrt{(x+y)^2 - 4xy}$

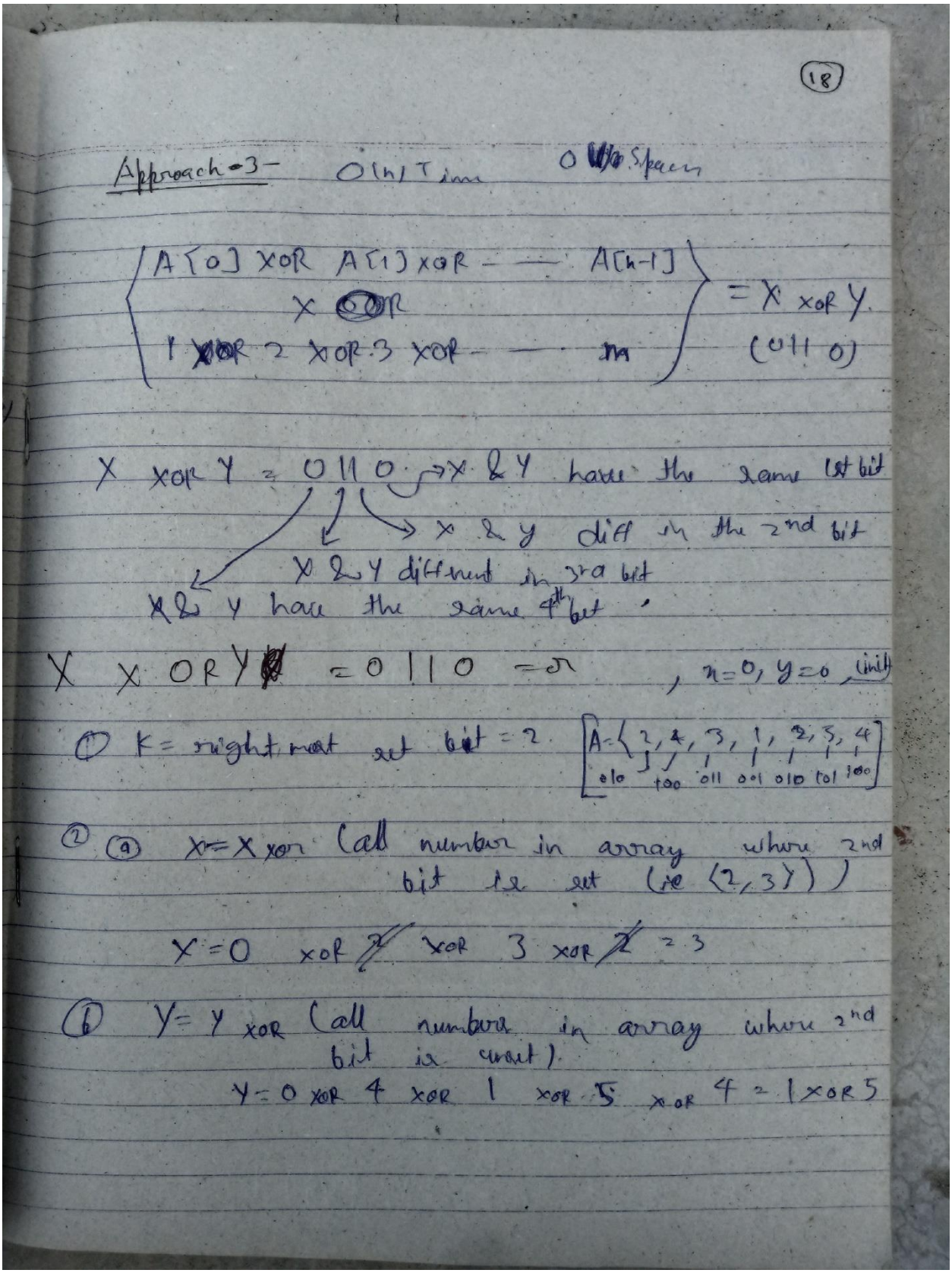
$(x-y) = \sqrt{(6)^2 - 4\times 8} = 2$

④ $x+y = 6$

$x-y = 2$

$2x = 8$

$x = 4$, by eq ① $y = 6-x = 2$


(18)
Approach-3- O(n) Time O(1) Space
A[0] XOR A[1] XOR ---- A[n-1]
X XOR
1 XOR 2 XOR 3 XOR ---- n
= X xor Y.
(011 0)
X xor Y = 011 0 → X & Y have the same 1st bit
X & y diff in the 2nd bit
X & Y different in 3rd bit
X & Y have the same 4th bit.
X XOR Y = 0 11 0 = 6 , n=0, y=0 (init)
① K = rightmost set bit = 2
A={2, 4, 3, 1, 2, 3, 4}
010 100 011 001 010 101 100
② (a) x=x xor (all number in array where 2nd bit is set (ie (2,3)))
X=0 xor 2 xor 3 xor 2 = 3
(b) Y=Y xor (all number in array where 2nd bit is unset)
Y=0 XOR 4 XOR 1 XOR 3 XOR 4 = 1 XOR 3

(19)

③ (a) X = X xor (all numbers in 1 to n with k^{th} bit set) → {2, 3}

X = X xor 2 xor 3 = 2

(b) Y = Y xor (all numbers in 1 to n with k^{th} bit unset) → {1, 4, 3}

Y = (X xor X) xor 1 xor 4 xor 3 = 4

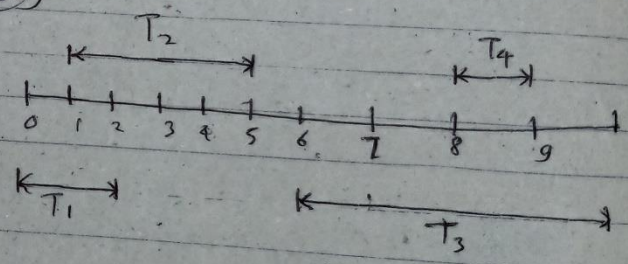
1.36 Merge Overlaping Intervals –

S = { T_1, T_2, T_3, T_4 } → n-time intervals

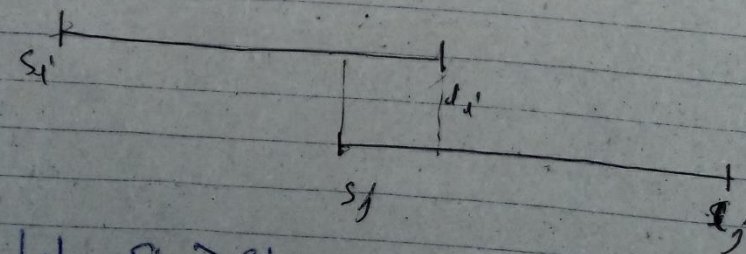
s_i f_i

T_1 = (0, 2)
T_2 = (1, 5)
T_3 = (6, 10)
T_4 = (8, 9)

(0, 5) (6, 10) return



When can we merge ? - T_i & T_j



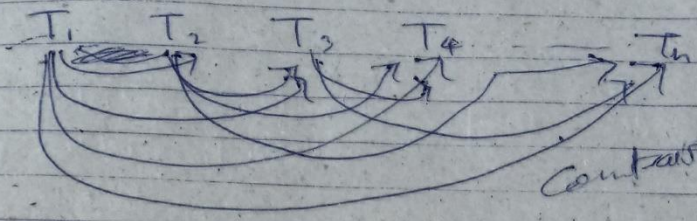
Let $s_j \geq s_i$

if $f_i' \geq s_j$

then (~~s_i~~, max(f_i', f_j))

(20)

Brute force Approach (n-intervals) - $O(n^2)$

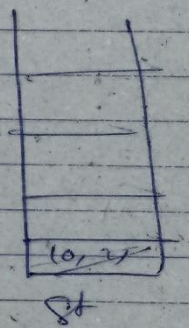


Approach-2 Tm: $O(n \log n)$, Sm: $O(n)$ use stack & sorting.

$S = \{T_1, T_2, T_3, T_4\}$

$T_1 = (s_1, e_1) = (0, 2)$
$T_2 = (1, 5)$
$T_3 = (6, 10)$
$T_4 = (8, 9)$

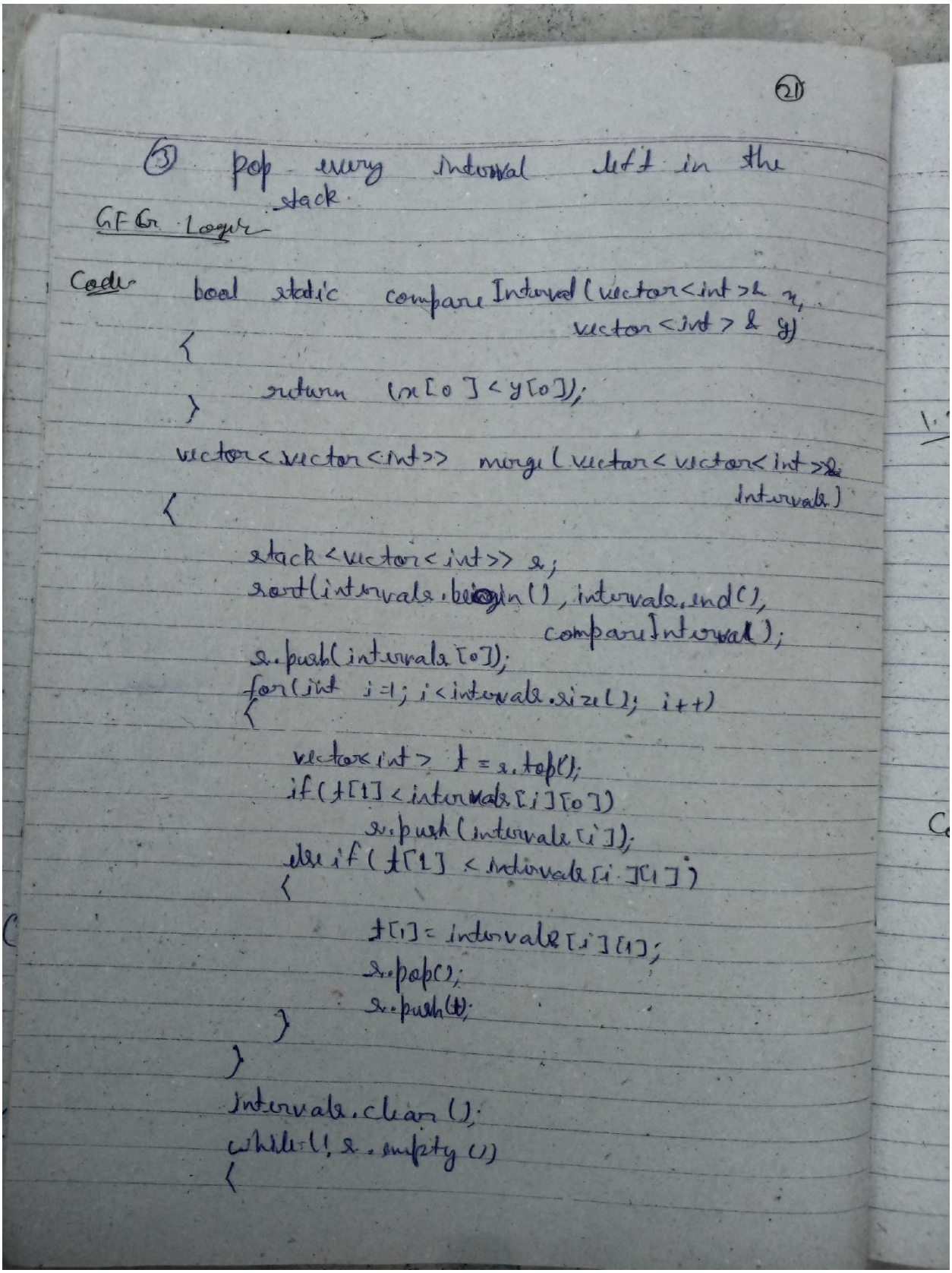
Sort according to S



① - Sort by s_i's

② for each T_j = 1 to n

```
if empty (St)
    push(Tj)
else
    Ti = Pop(St)
    if ei >= sj
        push(si, max(ei, ej))
    else
        push(Ti)
        push(Tj)
```



(21)

(3) pop every interval left in the stack.

GFG Logic

Code

```
bool static compareInterval(vector<int>& x,
                            vector<int>& y)
{
    return (x[0] < y[0]);
}
vector<vector<int>> merge(vector<vector<int>>&
                          intervals)
{
    stack<vector<int>> s;
    sort(intervals.begin(), intervals.end(),
         compareInterval);
    s.push(intervals[0]);
    for(int i=1; i<intervals.size(); i++)
    {
        vector<int> t = s.top();
        if(t[1] < intervals[i][0])
            s.push(intervals[i]);
        else if(t[1] < intervals[i][1])
        {
            t[1] = intervals[i][1];
            s.pop();
            s.push(t);
        }
    }
    intervals.clear();
    while(!s.empty())
    {
```

```
        vector<int> t = s.top();
        intervals.push_back(t);
        s.pop();
    }
    return intervals;
}
```

1.37 Rotate Matrix by 90° degree

I/P-

1	2	3
4	5	6
7	8	9

90° →

7	4	1
8	5	2
9	6	3

Transpose ↓

1	4	7
2	5	8
3	6	9

↑ reverse each row

Code-

```
void rotate(vector<vector<int>>& matrix)
{
    int n = matrix.size();
    for(int i=0; i<n; i++)
        for(int j=i+1; j<n; j++)
            swap(matrix[i][j], matrix[j][i]);
    for(int i=0; i<n; i++)
        for(int j=0; j<n/2; j++)
            swap(A[i][j], A[n-i+1, j])
```

(23)

1.39 3Sum-

Problem- Given an array nums of n integers, are there elements a, b, c in nums, such that a+b+c=0? Find all unique triplets in the array which give the sum of zero.
Notice that the solution set must not contain duplicate triplets.

I/P- [-1, 0, 1, 2, -1, -4]
O/P- [[-1, -1, 2], [-1, 0, 1]]

Approach-1 - Brute force O(n^3)

```
for i = 0 to n-2
    for j = i+1 to n-1
        for k = j+1 to n
            if A[i] + A[j] + A[k] == 0
                [ —— ]
```

Approach-2 Sorting + 2 pointer- O(n^2)

```
Sort arr.
for i = 0 to n-1
    l = i+1, r = n-1
    while (l < r)
        if sum == 0
            [ —— ]
        else if sum < 0
            l++
        else
            r--
```

(24)

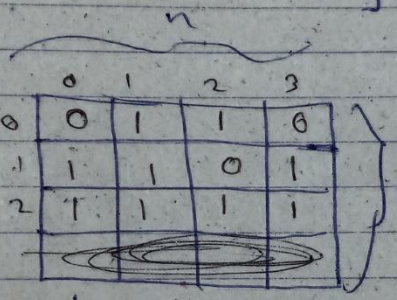
1.43 Set Matrix Zeroes -

Problem Statement - Given a matrix m x n, if an element is 0, set its entire row and column to 0. Do it in-place.

Ex- [[1, 1, 1], [1, 0, 1], [1, 1, 1]] ⇒ [[1, 0, 1], [0, 0, 0], [1, 0, 1]]

Approach

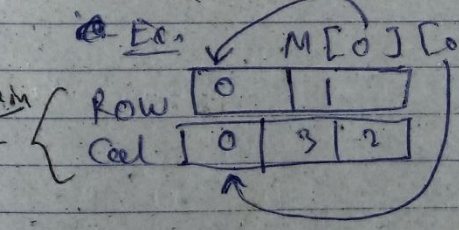
I/P -



~~T(n) = O(m*n)~~

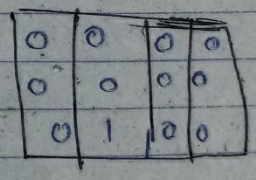
traverse and store row & col in a array

Ex. M[0][0] = 0



~~S(n) = O(m)~~

~~T(n) = O(m+n)~~

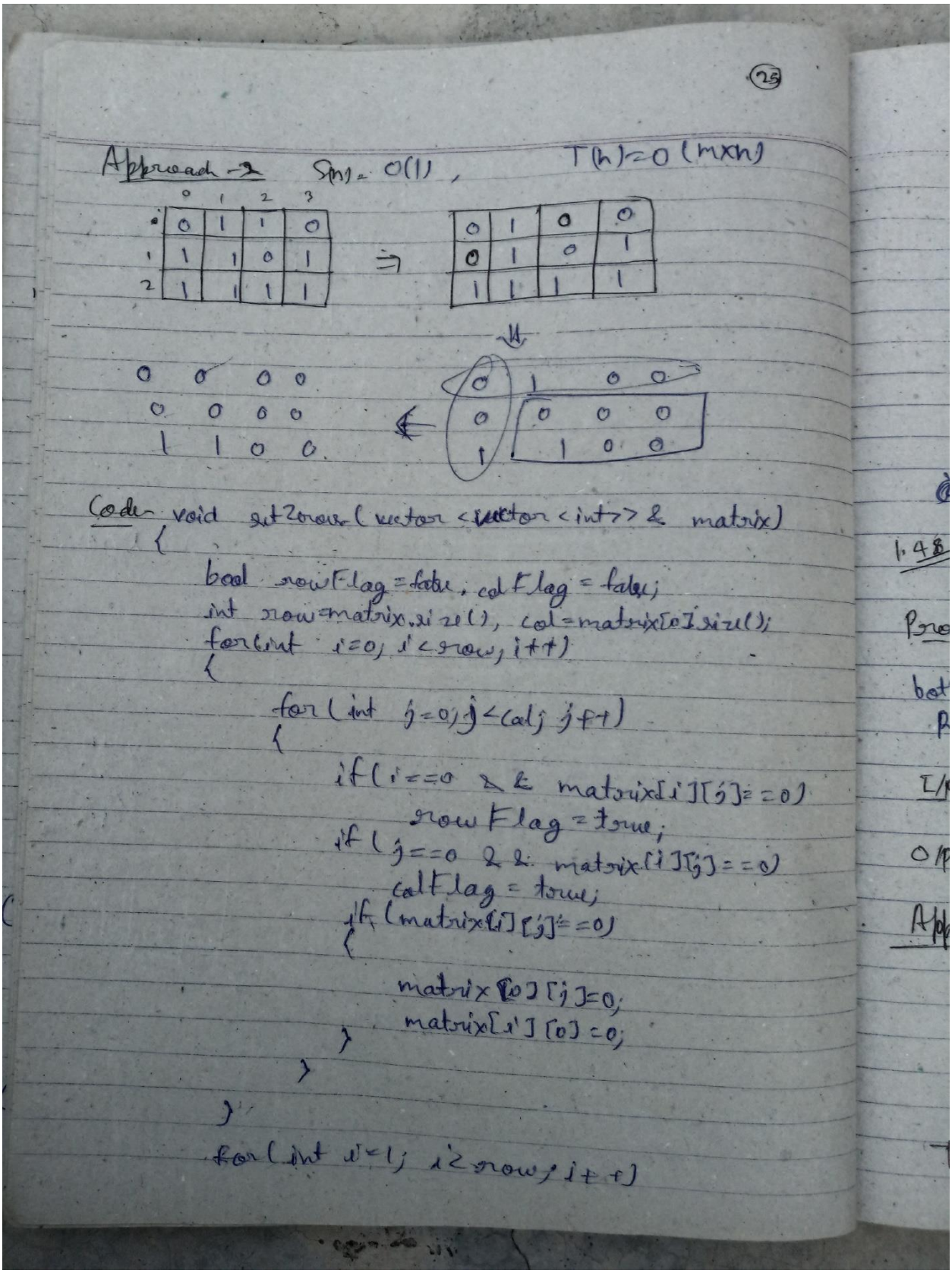


~~if i is row and~~

```
for i in row
    for j in col
        set M[i][j] = 0
```

T(n) = O(m*n)

S(n) = O(m+n)


25
Approach - 2 S(n) = O(1), T(n) = O(mxn)
Code
void setZeroes(vector<vector<int>>& matrix)
{
bool rowFlag = false, colFlag = false;
int row = matrix.size(), col = matrix[0].size();
for(int i=0; i<row; i++)
{
for(int j=0; j<col; j++)
{
if(i==0 && matrix[i][j]==0)
rowFlag = true;
if(j==0 && matrix[i][j]==0)
colFlag = true;
if(matrix[i][j]==0)
{
matrix[0][j]=0;
matrix[i][0]=0;
}
}
}
for(int i=1; i<row; i++)

```
for(j=0; j<n; j++)
    if( matrix[i][0]==0 || matrix[0][j]==0)
        matrix[i][j]=0;
if (rowFlag == true)
    for(int i=0; i<col; i++)
        matrix[0][i]=0;
if (colFlag == true)
    for(int j=0; j<row; j++)
        matrix[j][0]=0;
}
```

1.48 Count Negative Numbers in a Sorted Matrix -

Problem Statement - Given a m x n grid which is sorted in non-increasing order both row-wise and column-wise.
Return the no of negative numbers in grid -

I/p - grid = [[4,3,2,-1],[3,2,1,-1],[1,1,-1,-2],[-1,-1,-2,-3]]

O/p - 8

Approach-1

```
count = 0
for i=0 to row
    for j=0 to col
        if grid[i][j] < 0
            count++;
return count
```

T(N) = O(n*m)

(27)

Approach-2-

```
for (int i = r-1; i >= 0; i--)
    for( int j = 0 ; j < c ; j++)
        if (matrix[i][j] < 0)
        {
            count += (c - j);
            break;
        }
return count
```

4	3	2	1
3	2	1	1
1	1	-1	-2
-1	-1	-2	-3

1.5 The K weakest Row in a Matrix-

Problem- Given a m×n matrix mat of ones (representing soldiers) and zeros (representing civilians), return the indexes of k weakest rows in the matrix ordered form the weakest to the strongest.

A row i is weaker than j, if the number of soldiers in row i is less than the number of soldiers in row j, or they have the same number of soldiers but i is less than j. Soldiers are always stand in the frontier of a row, that is, always ones may appear first and then zeros.

Ex-

0	1	1	0	0	0 → 2
1	1	1	1	1	0 → 4
2	1	0	0	0	0 → 1
3	1	1	0	0	0 → 2
4	1	1	1	1	1 → 5

I/P → K=3

O/P - [2 | 0 | 3]

index

(28)

Approach: $T(n) = O(m \times n)$ $S(n) = O(m)$

	0	1	2	3	4	No. of 1's
0	1	1	0	0	0	2
1	1	1	1	1	0	4
2	1	0	0	0	0	1
3	1	1	0	0	0	2
4	1	1	1	1	1	5

(2,0) (4,1) (1,2) (2,3) (5,4) ⇒ Sort

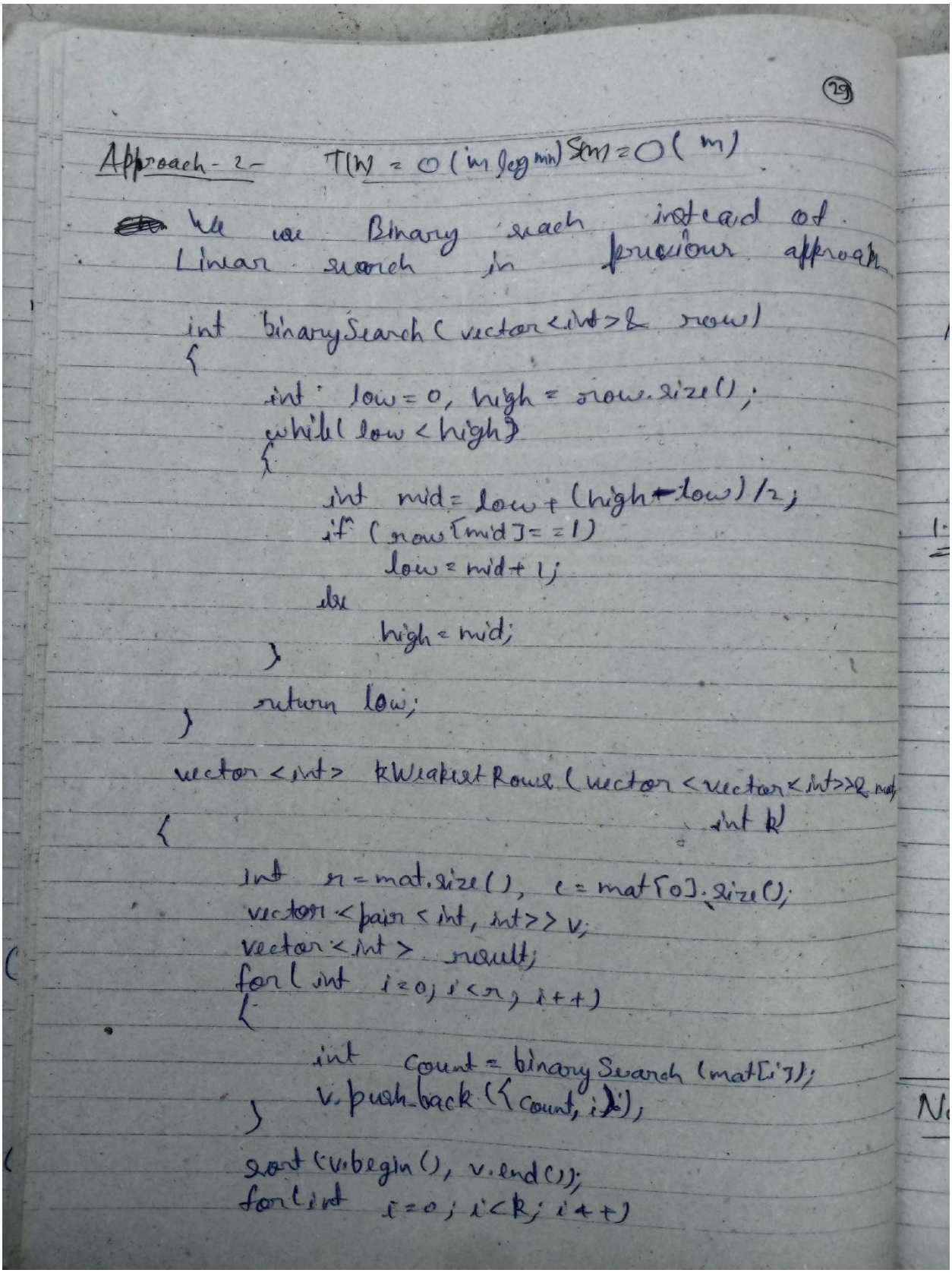
No. of 1's index

(1,2) (2,0) (2,3) (4,1) (5,4)

k=3

k-- return → [2, 0, 3]

```
vector<int> kWeakestRows(vector<vector<int>>& mat,
                                          int k)
{
    int r=mat.size(), c=mat[0].size();
    vector<int> result;
    for(int i=0; i<r; i++)
    {
        int count = accumulate(mat[i].begin(), mat[i]
                                          .end(), 0);
        v.push_back({count, i});
    }
    sort(v.begin(), v.end());
    for(int i=0; i<k; i++)
        result.push_back(v[i].second);
    return result;
}
```



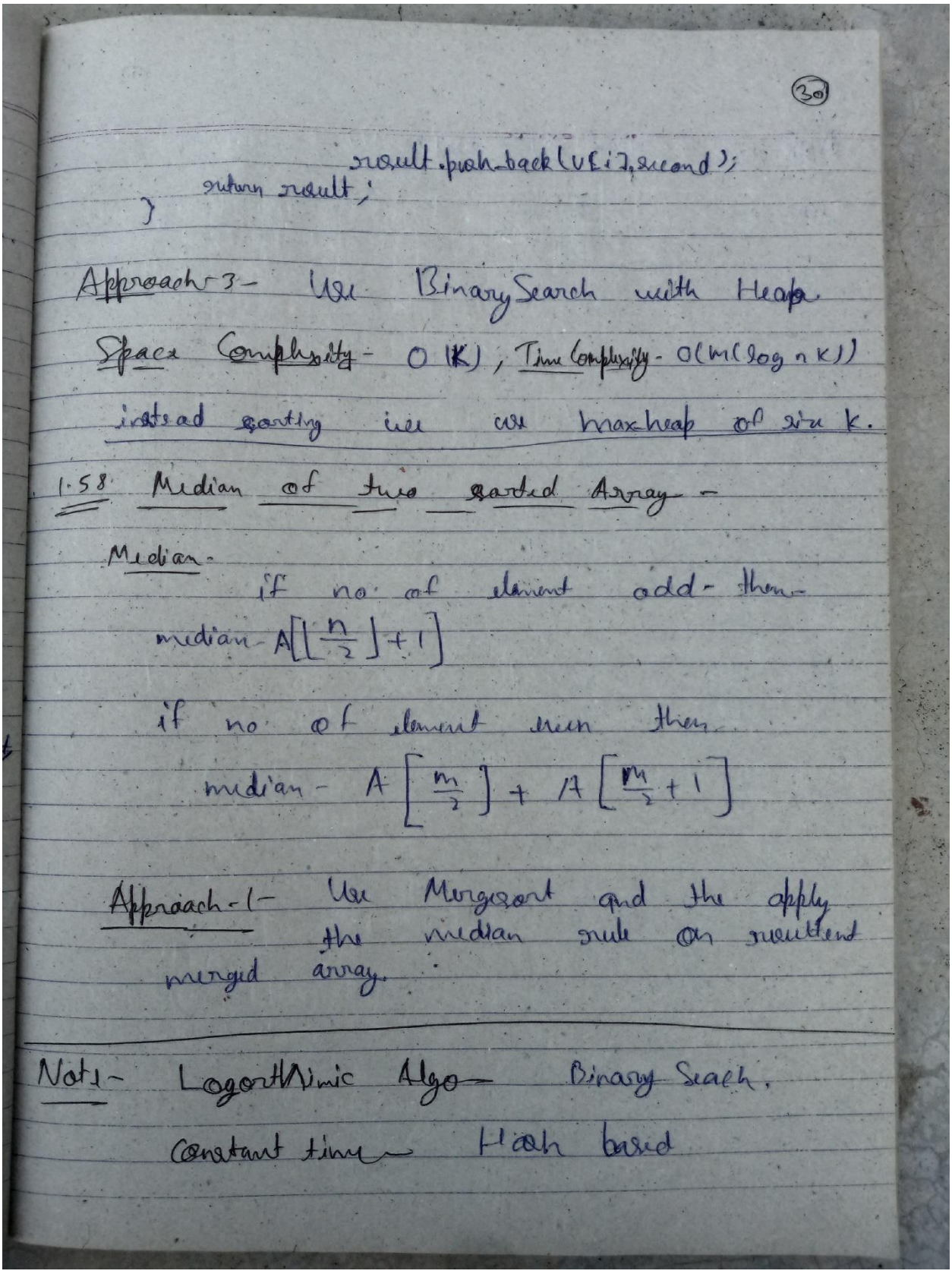
29

Approach - 2 - $T(N) = O(m \log m)$ $S(m) = O(m)$

We use Binary search instead of Linear search in previous approach.

```
int binarySearch(vector<int>& row)
{
    int low = 0, high = row.size();
    while(low < high)
    {
        int mid = low + (high - low)/2;
        if (row[mid] == 1)
            low = mid + 1;
        else
            high = mid;
    }
    return low;
}

vector<int> kWeakestRows(vector<vector<int>>& mat,
                         int k)
{
    int n = mat.size(), c = mat[0].size();
    vector<pair<int, int>> v;
    vector<int> result;
    for(int i=0; i<n; i++)
    {
        int count = binarySearch(mat[i]);
        v.push_back({count, i});
    }
    sort(v.begin(), v.end());
    for(int i=0; i<k; i++)
```



(30)

result.push_back(v[i], second);
}
return result;

Approach-3- Use BinarySearch with Heaps.

Space Complexity - O(K), Time Complexity - O(n(log nK))

instead sorting we use maxheap of size k.

1.58 Median of two sorted Array -

Median -

if no of element odd - then -

median - $A\left[\left[\frac{n}{2}\right]+1\right]$

if no of element even then -

median - $A\left[\frac{n}{2}\right] + A\left[\frac{n}{2}+1\right]$

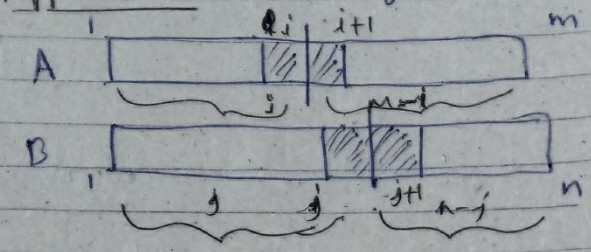
Approach-1- Use Mergesort and the apply the median rule on resulted merged array.

Note - Logorithmic Algo - Binary Seach.

Constant time - Hash based

(31)

Approach 2- T(n)=O(lg(max(m,n))) , S(n)=O(1)



Use Binary search to find middle element.

Find/search from j such that

1- $i+j = m+n-i-j$

2 (a) $A[i] < B[j+1]$

(b) $A[i+1] \geq B[j]$

1.59 First Missing Positive -

Given an unsorted integer array, find the smallest missing positive integer.

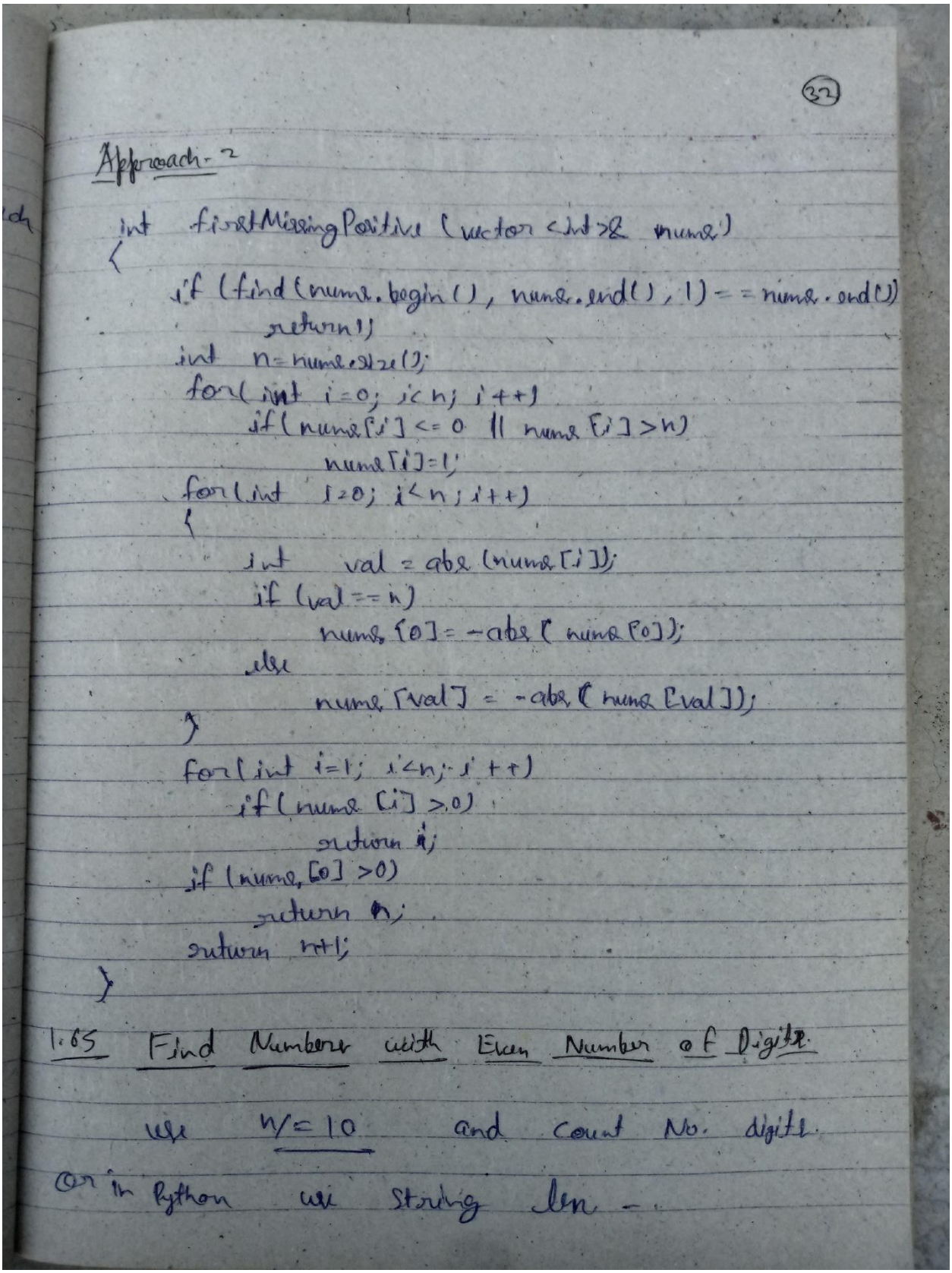
Ex- I/p - [1|2|0] O/p → 3

I/p - [3|4|-1|1] O/p → 2

I/p - [7|8|9|11|12] O/p → 1

Approach-1- Use Sorting - T(n)=O(nlgn) , S(n)=O(1)

```
int result = 1;
sort(nums.begin(), nums.end());
for(int i=0; i<nums.size(); i++)
    if (nums[i] == result)
        result++;
return result;
```



(32)

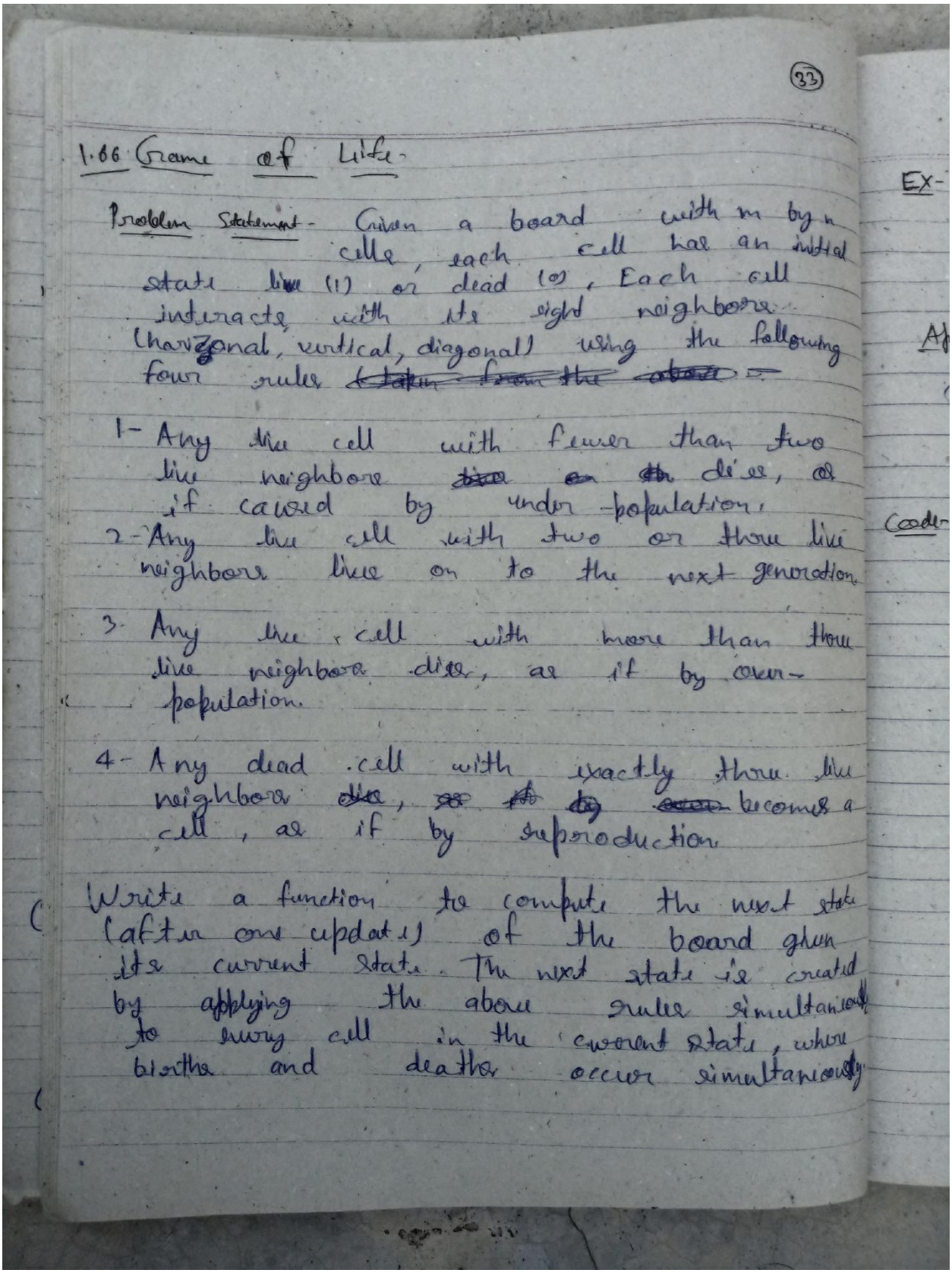
Approach-2

```
int firstMissingPositive(vector<int>& nums)
{
    if(find(nums.begin(), nums.end(), 1) == nums.end())
        return 1;
    int n=nums.size();
    for(int i=0; i<n; i++)
        if(nums[i]<=0 || nums[i]>n)
            nums[i]=1;
    for(int i=0; i<n; i++)
    {
        int val = abs(nums[i]);
        if(val==n)
            nums[0] = -abs(nums[0]);
        else
            nums[val] = -abs(nums[val]);
    }
    for(int i=1; i<n; i++)
        if(nums[i]>0)
            return i;
    if(nums[0]>0)
        return n;
    return n+1;
}
```

1.65 Find Numbers with Even Number of Digits

use n/=10 and count No. digits.

Or in Python use String len -.



(33)

1.66 Game of Life.

Problem Statement - Given a board with m by n cells, each cell has an initial state live (1) or dead (0). Each cell interacts with its eight neighbors (horizontal, vertical, diagonal) using the following four rules ~~(taken from the above)~~ :

1- Any live cell with fewer than two live neighbors ~~dies~~ dies, as if caused by under-population.

2- Any live cell with two or three live neighbors lives on to the next generation.

3. Any live cell with more than three live neighbors dies, as if by over-population.

4- Any dead cell with exactly three live neighbors ~~dies~~, ~~as if by~~ becomes a cell, as if by reproduction.

Write a function to compute the next state (after one update) of the board given its current state. The next state is created by applying the above rules simultaneously to every cell in the current state, where births and deaths occur simultaneously.

(34)

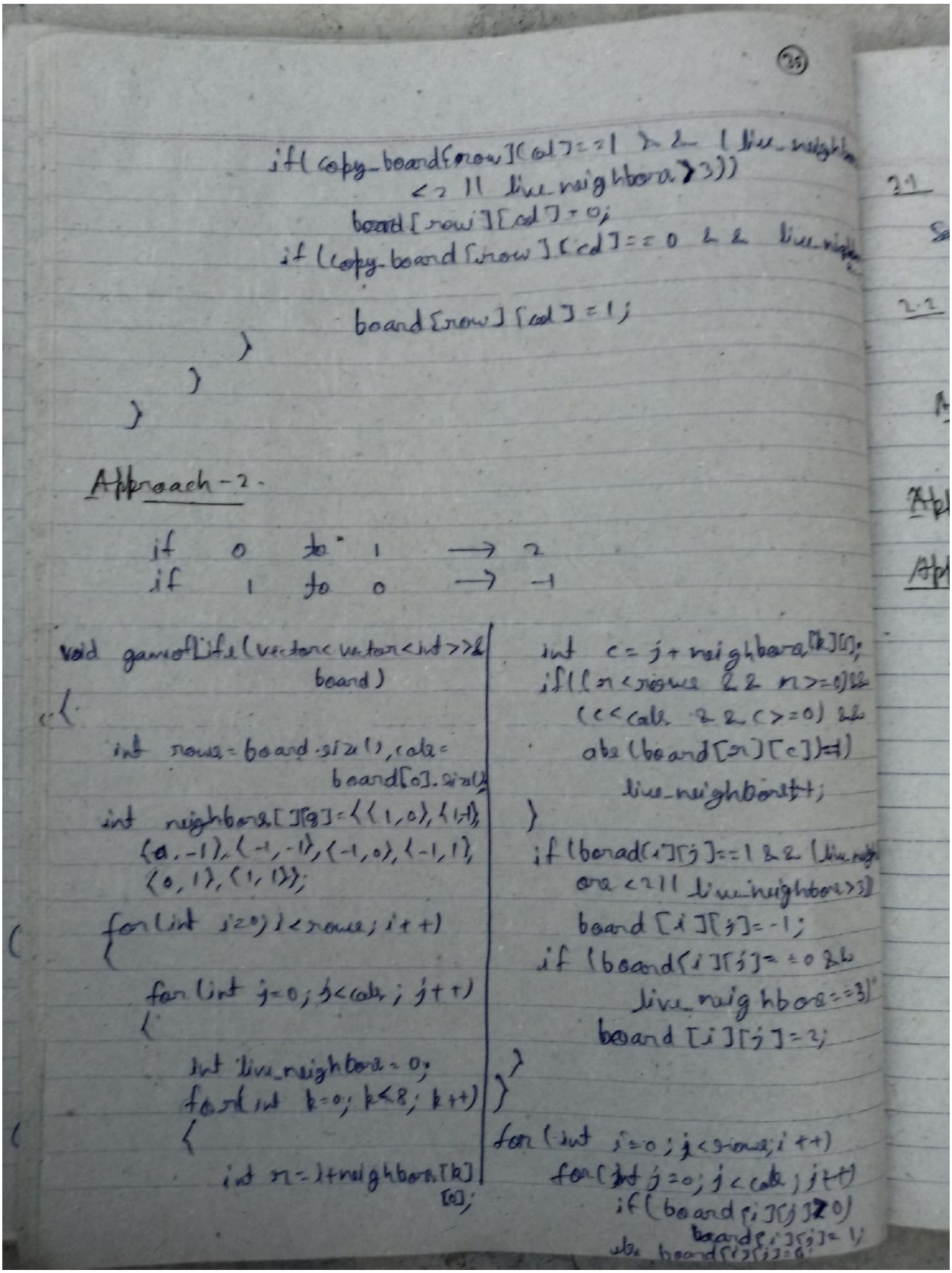
Ex- [[0, 1, 0], [0, 0, 1], [1, 1, 1], [0, 0, 0]] ⇒ [[0, 0, 0], [1, 0, 1], [0, 1, 1], [0, 1, 0]]

Approach-1 — T(n) = O(m*n), S(n) = O(m*n)

Create new copy of array and then check all 4 condition for each element using copy of array and then modify original array according to that.

Code-

```
void gameOfLife(vector<vector<int>>& board)
{
    int rows = board.size(), cols = board[0].size();
    int neighbors = {{1,0},{1,-1},{0,-1},{-1,-1},{-1,0},
                     {-1,1},{0,1},{1,1}};
    vector<vector<int>> copy_board(board);
    for(int i=0; i<rows; i++)
    {
        for(int j=0; j<cols; j++)
        {
            int liveneighbors=0;
            for(int k=0; k<8; k++)
            {
                int r = i+neighbors[k][0];
                int c = j+neighbors[k][1];
                if((r<rows and r>=0) && (c<cols
                    and copy_board[r][c]==1)
                liveneighbors++;
            }
```



```
        if(copy_board[row][col]==1 && (live_neighbors
                <2 || live_neighbors>3))
            board[row][col]=0;
        if(copy_board[row][col]==0 && live_neighbors
            board[row][col]=1;
        }
    }
}
```

Approach-2.

if 0 to 1 ⟶ 2
if 1 to 0 ⟶ -1

```
void gameOfLife(vector<vector<int>>&
                board)
{
    int rows=board.size(), cols=
                board[0].size();
    int neighbors[][8]={{1,0},{1,1},
    {0,-1},{-1,-1},{-1,0},{-1,1},
    {0,1},{1,1}};
    for(int i=0;i<rows;i++)
    {
        for(int j=0;j<cols;j++)
        {
            int live_neighbors=0;
            for(int k=0;k<8;k++)
            {
                int r=i+neighbors[k]
                [0];
                int c=j+neighbors[k][1];
                if((r<rows && r>=0)&&
                (c<cols && c>=0) &&
                abs(board[r][c])==1)
                    live_neighbors++;
            }
            if(board[i][j]==1 && (live_neigh
                bors<2||live_neighbors>3))
                board[i][j]=-1;
            if(board[i][j]==0 &&
                live_neighbors==3)
                board[i][j]=2;
        }
    }
    for(int i=0;i<rows;i++)
        for(int j=0;j<cols;j++)
            if(board[i][j]>0)
                board[i][j]=1;
            else board[i][j]=0;
```

Problems on Searching and Sorting

(36)

2.1 Sort an array of 0's, 1's and 2's -

Same as - 1.30

2.2 K'th Smallest / Largest Element in Unsorted Array -

Approach 1 - Use Sorting and return K th smallest / Largest - $T(n) = O(n \lg n)$, $S(n) = O(n)$

Approach 2 - Use Heap - , $T(n) = O(n + k \lg n)$.

Approach 3 - Use Quick Select - $O(n)$.

```
int partition(vector<int>& nums, int start, int end)
{
    int pivot = nums[end];
    int count = start;
    for(int i=start; i<end; i++)
    {
        if (nums[i] > pivot)
        {
            swap(nums[i], nums[count]);
            count++;
        }
    }
    swap(nums[count], nums[end]);
    return count;
}
if findKthLargest(vector<int>& nums, int k)
{
```

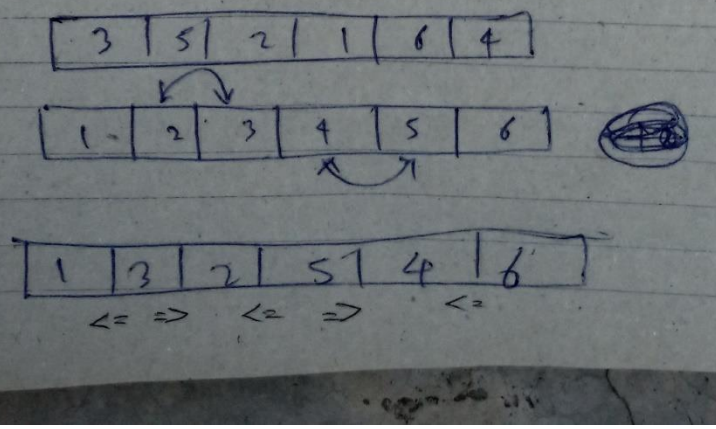
(37)

```
int start = 0, end = nums.size()-1;
while(start <= end)
{
    int par = partition(nums, start, end);
    if(par == k-1)
        return nums[par];
    else if(par > k-1)
        end = par-1;
    else
        start = par+1;
}
return -1;
}
```

2.3 Wiggle Sort - Given an unsorted array, nums, reorder it in-place such that nums[0] <= nums[1] >= nums[2] <= nums[3] ....

Ex- I/P - nums = [3, 5, 2, 1, 6, 4]
(0 1 2 3 4 5)
O/P - one possible O/P - [3, 5, 1, 6, 2, 4]

Approach-1 Sorting. T(n)=O(nlgn) S(n)=O(1)



Code-
```
void wiggleSort(vector<int>& v)
{
    sort(v.begin(), v.end());
    for(int i=1; i<v.size()-1; i+=2)
        swap(v[i], v[i+1]);
}
```

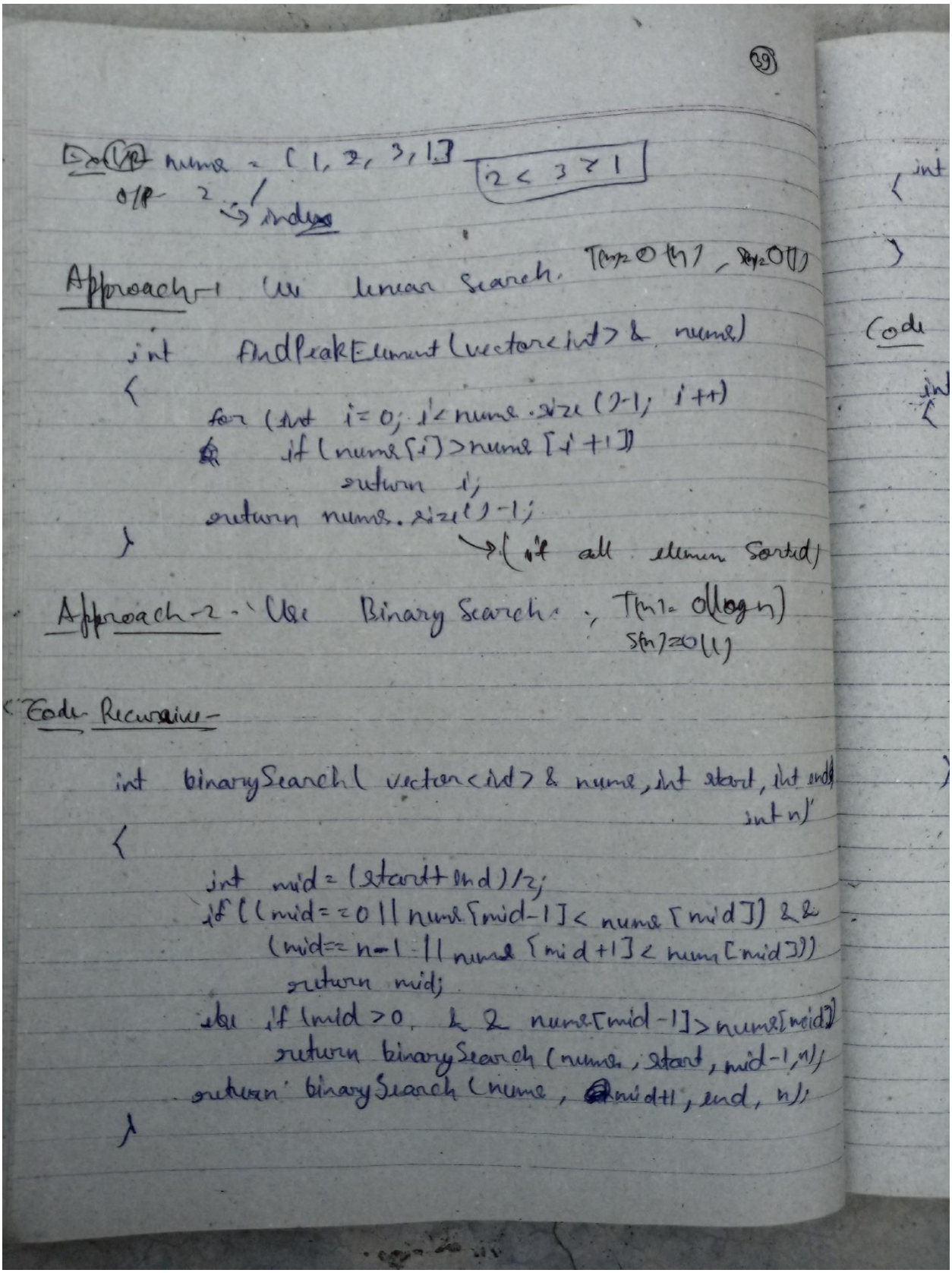
Approach-2 : T(n) = O(n) S(n) = O(1)

pattern- 1st <= 2nd => 3rd <= 4th => 5th ----

Traverse array only visiting even position ele.

Code-
```
void wiggleSort(vector<int>& v)
{
    for(int i=1; i<v.size(); i+=2)
    {
        if(i>0 && v[i]<v[i-1])
            swap(v[i], v[i-1]);
        if(i<v.size()-1 && v[i]<v[i+1])
            swap(v[i], v[i+1]);
    }
}
```

28 Find Peak Element - A peak element is an element that is strictly greater than its neighbors. Given an integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks.



39

Ex- nums = [1, 2, 3, 1]
O/P - 2 → Index
2 < 3 > 1

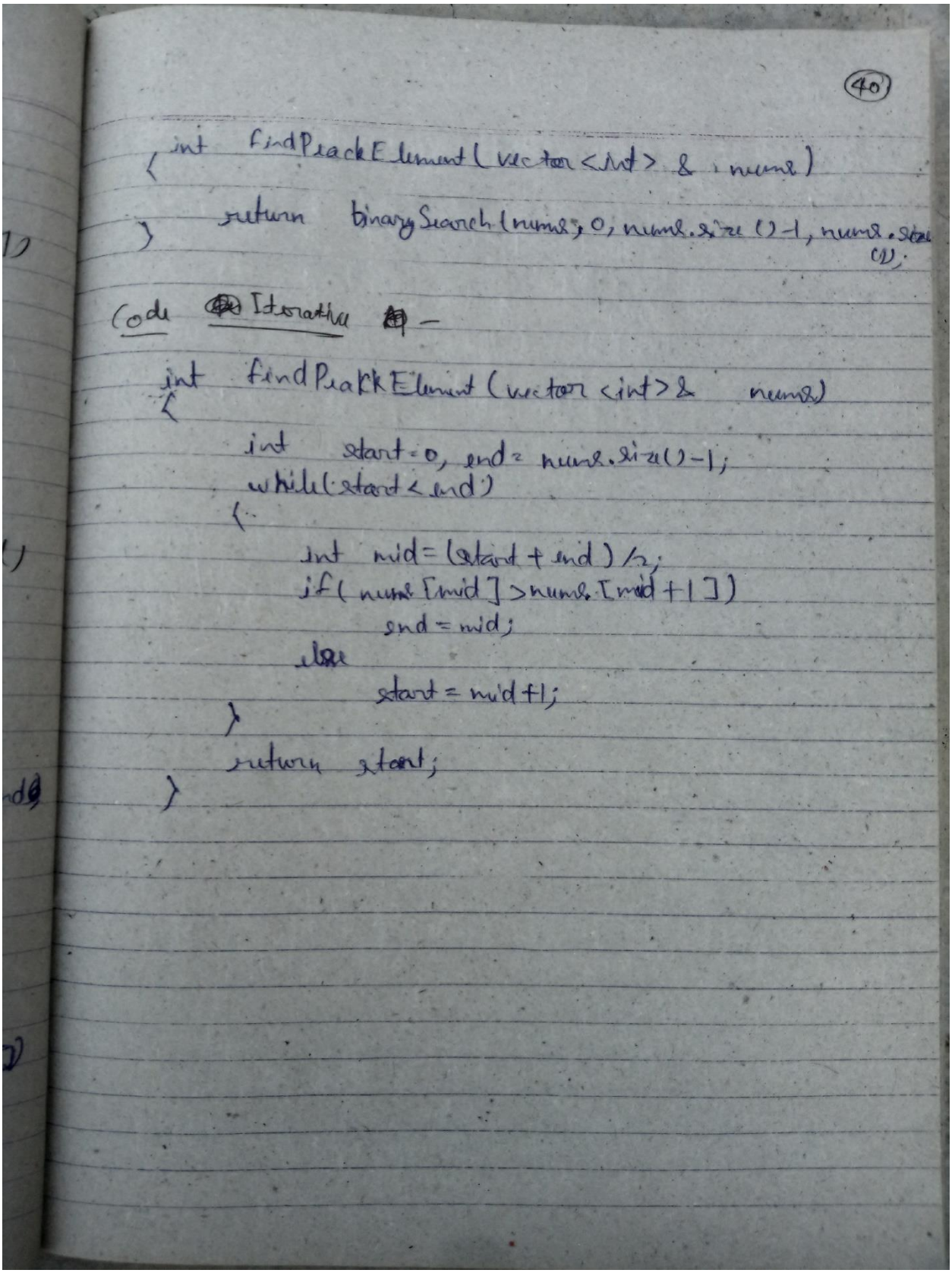
Approach-1 (Using linear Search) T(n) = O(n), S(n) = O(1)

```
int findPeakElement(vector<int>& nums)
{
    for (int i=0; i<nums.size()-1; i++)
        if (nums[i] > nums[i+1])
            return i;
    return nums.size()-1;
}
```
→ (if all elements sorted)

Approach-2 - Use Binary Search, T(n) = O(log n), S(n) = O(1)

Code Recursive -

```
int binarySearch(vector<int>& nums, int start, int end, int n)
{
    int mid = (start+end)/2;
    if ((mid==0 || nums[mid-1] < nums[mid]) &&
        (mid==n-1 || nums[mid+1] < nums[mid]))
        return mid;
    else if (mid>0 && nums[mid-1] > nums[mid])
        return binarySearch(nums, start, mid-1, n);
    return binarySearch(nums, mid+1, end, n);
}
```



(40)

```
int findPeakElement(vector<int>& nums)
{
    return binarySearch(nums, 0, nums.size()-1, nums.
                                                  ());
}
```

Code (Iterative) -

```
int findPeakElement(vector<int>& nums)
{
    int start=0, end= nums.size()-1;
    while(start<end)
    {
        int mid=(start+end)/2;
        if(nums[mid]>nums.[mid+1])
            end=mid;
        else
            start=mid+1;
    }
    return start;
}
```

Problems on Linked List (41)

3.1- Find Kth node from end of a linked list -

Approach-1- $T(n) = O(n)$, $S(n) = O(1)$

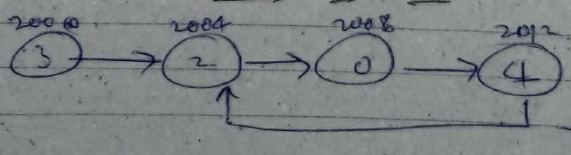
1) Count length of the list
2) return (length - K)th node

Approach-2- $T(n) = O(n)$ $S(n) = O(1)$

1) p & q → start
2) Move q to k^{th} node from begining
3) Move p & q node step by step in random till q reaches the last node

→ p reaches $(n-k)^{th}$ from begin = k^{th} node from end

3.2- Linked List Cycle-



Approach-1- $T(n) = O(n)$ $S(n) = O(n)$

Traverse list and.
- Use Hash table to store each node address and if any address already in Hashtable then there is a cycle otherwise there is no any cycle

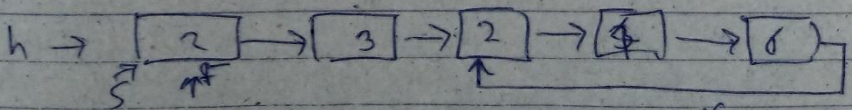
(42)

Code-

```
bool hasCycle(ListNode *head)
{
    if (head==NULL || head->next==NULL)
        return false;
    unordered_map<struct ListNode*, bool> m;
    struct ListNode *ptr=head;
    while(ptr)
    {
        if(m[ptr]==false)
        {
            m[ptr]=true;
            ptr=ptr->next;
        }
        else
            return true;
    }
    return false;
}
```

Approach 2 - $T(n)=O(n)$ $S(n)=O(1)$

Floyd's Cycle detection Algo. or Hare & Tortoise or Fast Slow Algo



iteration
S: move by 1 step
F: move by 2 steps

if(F==S) have loop

if S→NULL
F→NULL
F→next=NULL
have No Loop

(43)

Code

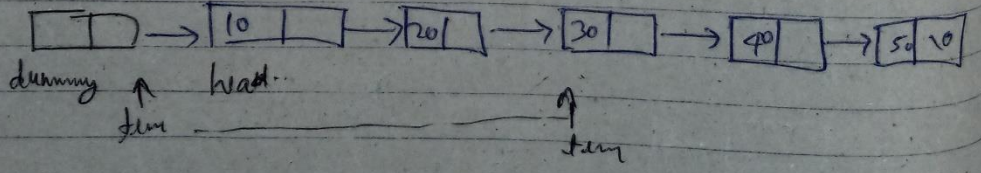
```
bool hasCycle(ListNode *head)
{
    struct ListNode *fastPtr = head, *slowPtr = head;
    while(fastPtr && fastPtr->next)
    {
        fastPtr = fastPtr->next->next;
        slowPtr = slowPtr->next;
        if(fastPtr == slowPtr)
            return true;
    }
    return true;
}
```

3.3 Remove Nth node from End of linked List

Approach 1- T(n) = O(n) S(n) = O(1)

1) Count Length of Linked List

2) Traverse the list upto $(L-n)^{th}$ node from starting.

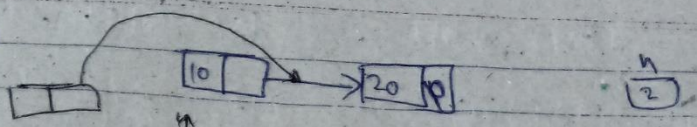


temp -> next = temp -> next -> next;

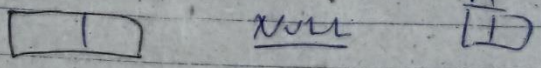
Why do we need dummy node?
To handle the corner case.

(44)

(i)



(ii)



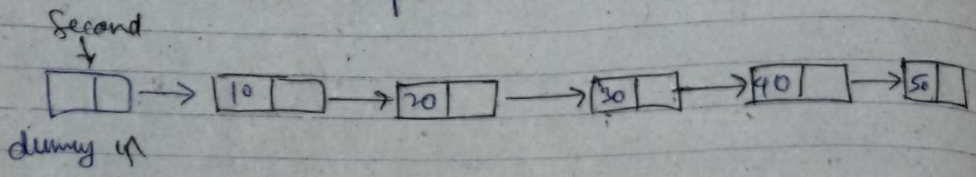
Code -

```
ListNode * removeNthFromEnd(ListNode *
                            head, int n)
{
    int length = 0;
    struct ListNode *dummy, *temp = head;
    dummy = (struct ListNode*) malloc(sizeof(struct
                                      ListNode));
    dummy -> val = 0;
    dummy -> next = head;
    while (temp)
    {
        length++;
        temp = temp -> next;
    }
    length -= n;
    temp = dummy;
    while (length--)
        temp = temp -> next;
    temp -> next = temp -> next -> next;
    return dummy -> next;
}
```

45

Approach 2 - $T(n) = O(n)$ $S(n) = O(1)$

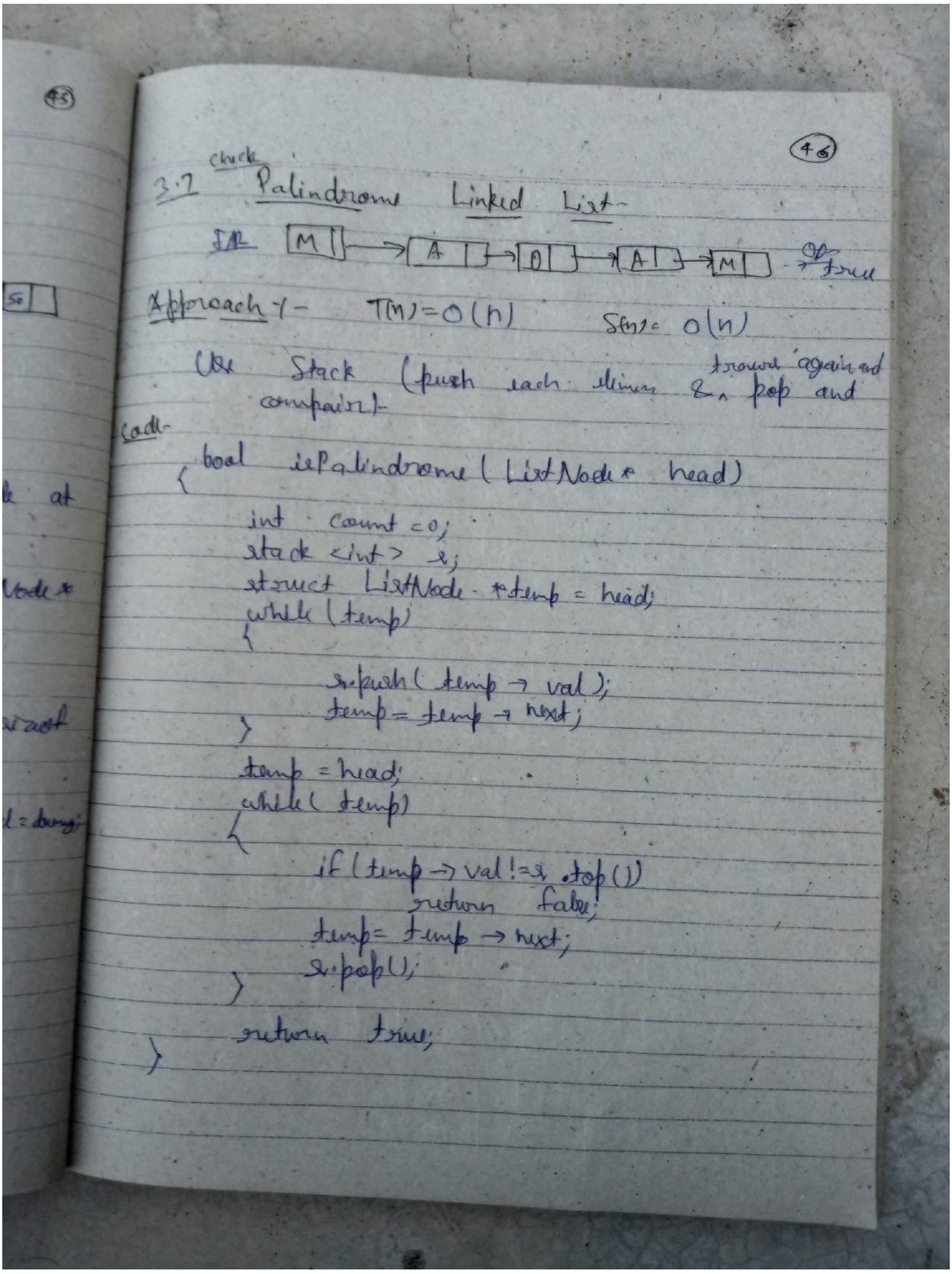
Use two pointer Approach



1) Move first ptr upto n nodes
2) Move both the pointer one node at a time

Code -

```
ListNode* removeNthNodeFromEnd(ListNode* head, int n)
{
    struct ListNode *dummy;
    dummy = (struct ListNode*) malloc(sizeof(struct ListNode));
    dummy->next = head;
    struct ListNode *first = dummy, *second = dummy;
    for(int i=0; i<=n; i++)
        first = first->next;
    while(first)
    {
        first = first->next;
        second = second->next;
    }
    second->next = second->next->next;
    return dummy->next;
}
```



3.7 check Palindrome Linked List-

I/P: M → A → D → A → M → O/P: true

Approach :- T(n) = O(n) S(n) = O(n)

Use Stack (push each element & pop and traverse again and compare)-

code-

```
bool isPalindrome(ListNode* head)
{
    int count = 0;
    stack<int> s;
    struct ListNode *temp = head;
    while(temp)
    {
        s.push(temp->val);
        temp = temp->next;
    }
    temp = head;
    while(temp)
    {
        if(temp->val != s.top())
            return false;
        temp = temp->next;
        s.pop();
    }
    return true;
}
```

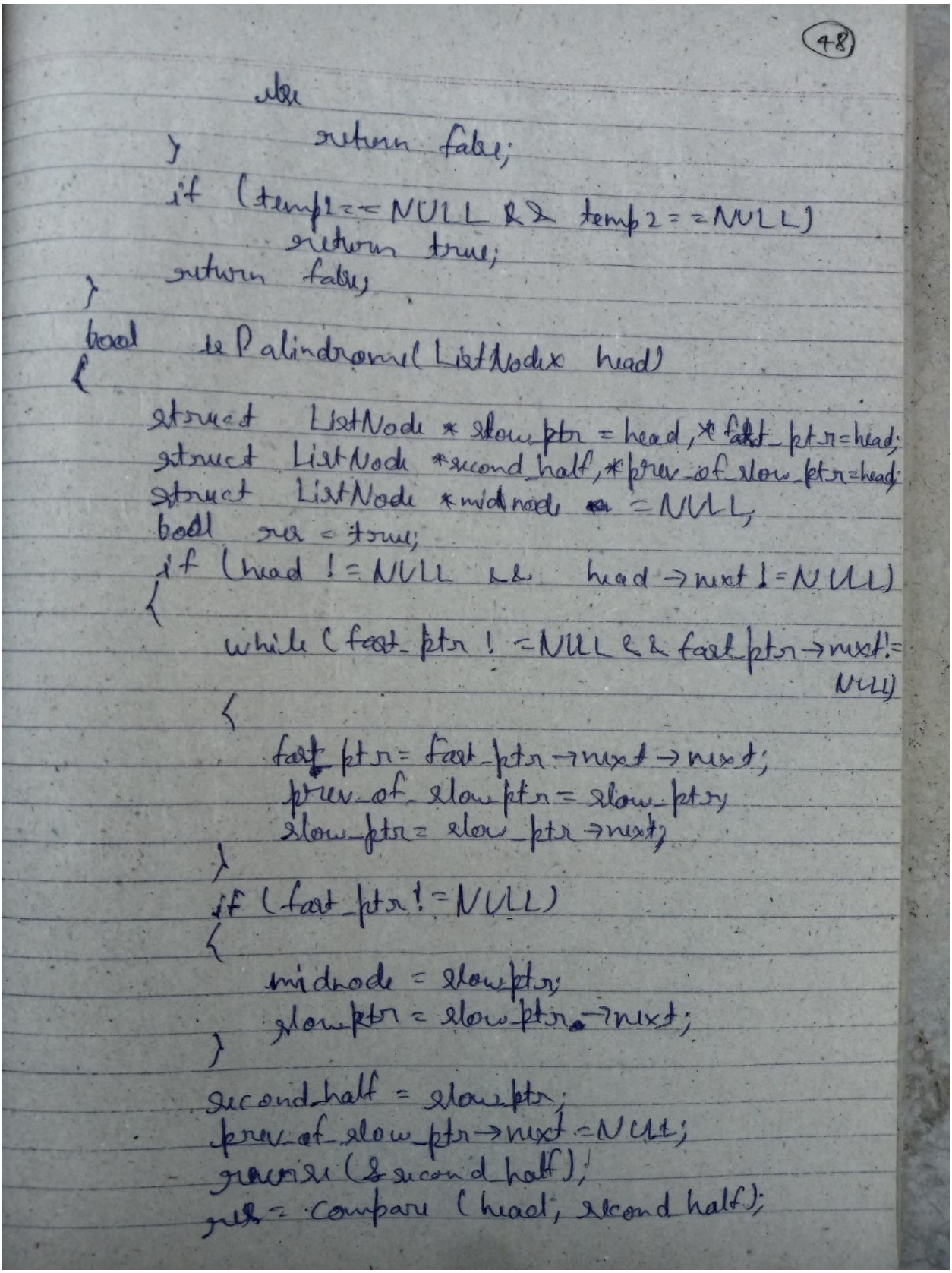
(47)

Approach-2 - $T(n) = O(n)$ $S(n) = O(1)$

1) reach the middle of the L.L. → Count. → Slow, Fast
2) reverse the 2nd half of the L.L
3) Compare
4) Reverse the 2nd half back to normal.

Code-

```
void reverse (struct ListNode ** head)
{
    struct ListNode *prev = NULL;
    struct ListNode * current = *head;
    struct ListNode * next;
    while (current != NULL)
    {
        next = current -> next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head=prev;
}
bool compare (struct ListNode* temp1, struct
                                ListNode* temp2)
{
    while (temp1 && temp2)
    {
        if (temp1 -> val == temp2 -> val)
        {
            temp1 = temp1 -> next;
            temp2 = temp2 -> next;
        }
```

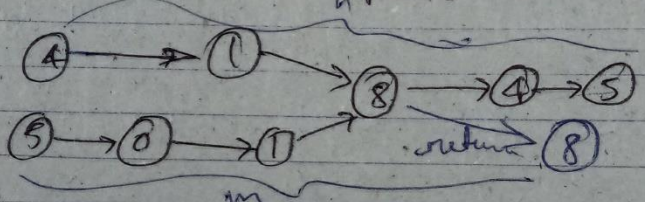


```
(48)

        else
            return false;
    }
    if (temp1 == NULL && temp2 == NULL)
        return true;
    return false;
}
bool isPalindrome(ListNode* head)
{
    struct ListNode * slow_ptr = head, * fast_ptr = head;
    struct ListNode *second_half, *prev_of_slow_ptr = head;
    struct ListNode *midnode = NULL;
    bool res = true;
    if (head != NULL && head->next != NULL)
    {
        while (fast_ptr != NULL && fast_ptr->next !=
                                                NULL)
        {
            fast_ptr = fast_ptr->next->next;
            prev_of_slow_ptr = slow_ptr;
            slow_ptr = slow_ptr->next;
        }
        if (fast_ptr != NULL)
        {
            midnode = slow_ptr;
            slow_ptr = slow_ptr->next;
        }
        second_half = slow_ptr;
        prev_of_slow_ptr->next = NULL;
        reverse(&second_half);
        res = compare(head, second_half);
```

```
reverse(&second_half);
if (midnode != NULL)
{
    prevof_slow_ptr->next = midnode;
    midnode->next = second_half;
}
else
    prevof_slow_ptr->next = second_half;
}
return res;
}
```

3.9 Intersection point of Two Linked List-

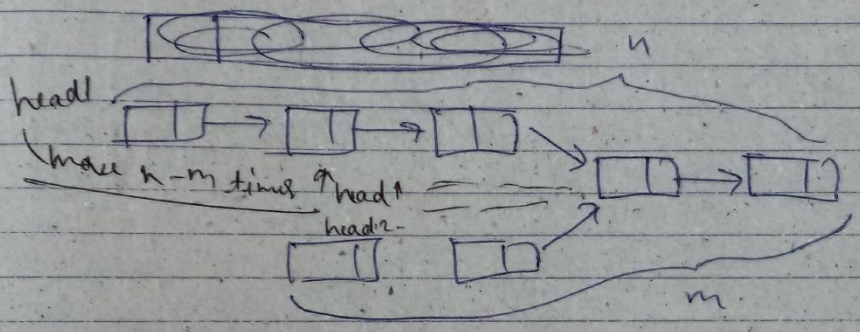


Approach I- T(n) = O(n*m) S(n) = O(1)

```
ListNode *getIntersectionNode(ListNode *headA, ListNode *headB)
{
    ListNode *temp1 = headA, *temp2 = headB;
    while(temp1)
    {
        temp2 = headB;
        while(temp2)
        {
            if(temp1 == temp2)
                return temp1;
```

```
        temp2 = temp2->next;
    }
    temp1 = temp1->next;
}
return NULL;
}
```

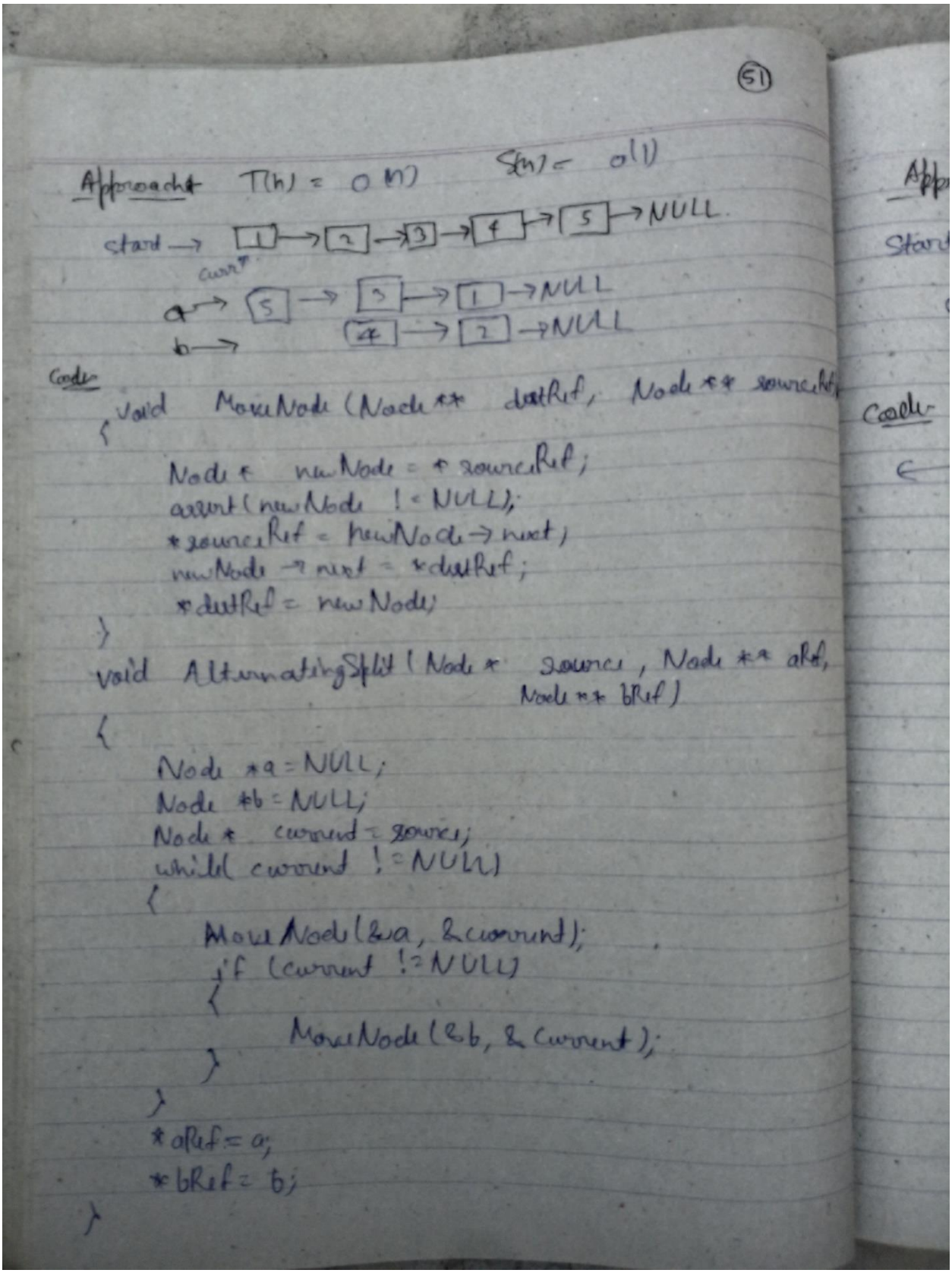
Approach 2- $T(n) = O(n)$ $S(n) = O(1)$



3.11 Alternative split of singly Linked List-

Problem Statement- Write a function AlternatingSplit that take one list and divides up its nodes to make two smaller lists 'a' and 'b'. The sublists should be made from alternating elements in the original list. So if the original list is -

$0 \to 1 \to 0 \to 1 \to 0 \to 1$ then one sublist should be $0 \to 0 \to 0$ and the other should be $1 \to 1 \to 1$.



(51)

Approach: T(n) = O(n) S(n) = O(1)

start → 1 → 2 → 3 → 4 → 5 → NULL

curr

a → 5 → 3 → 1 → NULL

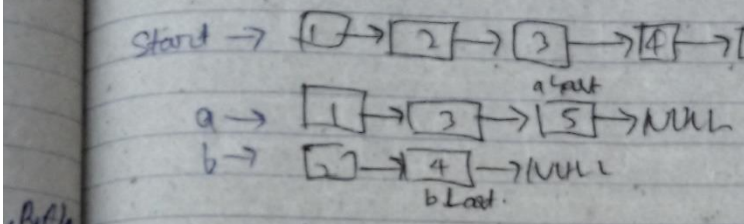
b → 4 → 2 → NULL

Code

```
void MoveNode(Node** destRef, Node** sourceRef)
{
    Node* newNode = *sourceRef;
    assert(newNode != NULL);
    *sourceRef = newNode->next;
    newNode->next = *destRef;
    *destRef = newNode;
}
void AlternatingSplit(Node* source, Node** aRef,
                      Node** bRef)
{
    Node *a = NULL;
    Node *b = NULL;
    Node* current = source;
    while(current != NULL)
    {
        MoveNode(&a, &current);
        if (current != NULL)
        {
            MoveNode(&b, &current);
        }
    }
    *aRef = a;
    *bRef = b;
}
```

(52)

Approach 2 - T(n) = O(n) S(n) = O(1)



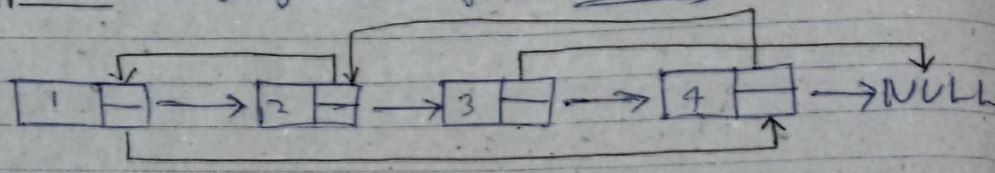
Code: void MoveNode(Node** destRef, Node** sourceRef)
{
← Same }

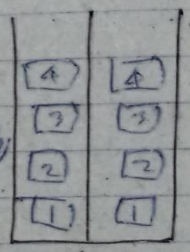
```
void AlternatingSplit(Node* source, Node** aRef,
                      Node** bRef)
{
    Node aDummy;
    Node* aTail = &aDummy;
    Node bDummy;
    Node* bTail = &bDummy;
    Node* current = source;
    aDummy.next = NULL;
    bDummy.next = NULL;
    while(current != NULL)
    {
        MoveNode(&(aTail->next), &current);
        aTail = aTail->next;
        if(current != NULL)
        {
            MoveNode(&(bTail->next), &current);
            bTail = bTail->next;
        }
    }
    *aRef = aDummy.next;
    *bRef = bDummy.next;
}
```

53

3.15 Clone List with Random Pointer-

Approach 1- Using HashMap- Sw = O(n), T(n) = O(n)





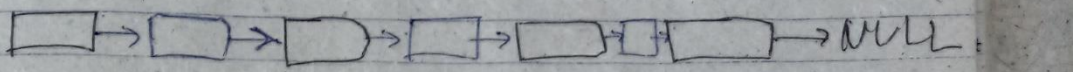
<Node*, Node*>

Code-

```
Node *copyRandomList(Node *head)
{
    unordered_map<Node*, Node*> nodeMap;
    if (!head)
        return NULL;
    Node * dummy = new Node(0);
    Node * curnew = dummy;
    Node *cur=head;
    while(cur)
    {
        curnew->next = new Node(cur->val);
        nodeMap[cur] = curnew->next;
        curnew= curnew->next;
        cur= cur->next;
    }
    curnew = dummy->next;
    cur=head;
    while(cur)
    {
        curnew->random=nodeMap[cur->random];
        curnew=curnew->next;
        cur= cur->next;
    }
    return dummy->next;
}
```

(54)

Approach 2- T(n) = O(n) S(n) = O(1).



1) creating new nodes inbetween the old nodes.

2) Curr → next → prev = curr → prev → next;
   curr = curr → next → next;

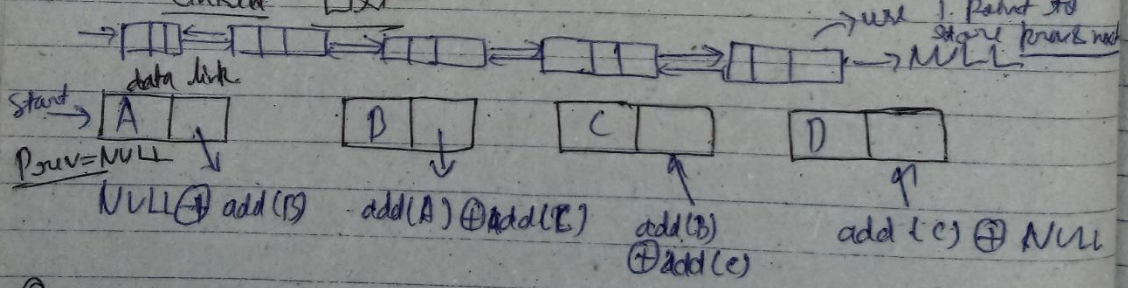
3) Split this DLL into 2 parts → original
                              → copy

```
Code- Node* copyRandomList(Node * head)
{
    if (!head)
        return NULL;
(1)→ Node *curr = head, *temp;
    while (curr)
    {
        temp = curr → next;
        Node * temp1 = new Node(0);
        temp1 → val = curr → val;
        temp1 → random = NULL;
        curr → next = temp1;
        curr → next → next = temp;
        curr = temp;
    }
];
    curr = head;
(2)→ while (curr)
    {
        if (curr → next)
```

55

```
curr->next->random = curr->random ?
                curr->random->next : curr->random;
curr = curr->next ? curr->next->next : curr->next;
}
Node *original = head; *copy = head->next;
temp = copy;
while(original && copy)
{
original->next = original->next ? original->next->next :
                                  original->next;
copy->next = copy->next ? copy->next->next : original->next;
copy = copy->next;
}
return temp;
}
```

3.16 XOR Linked List – A Memory Efficient Doubly Linked List –

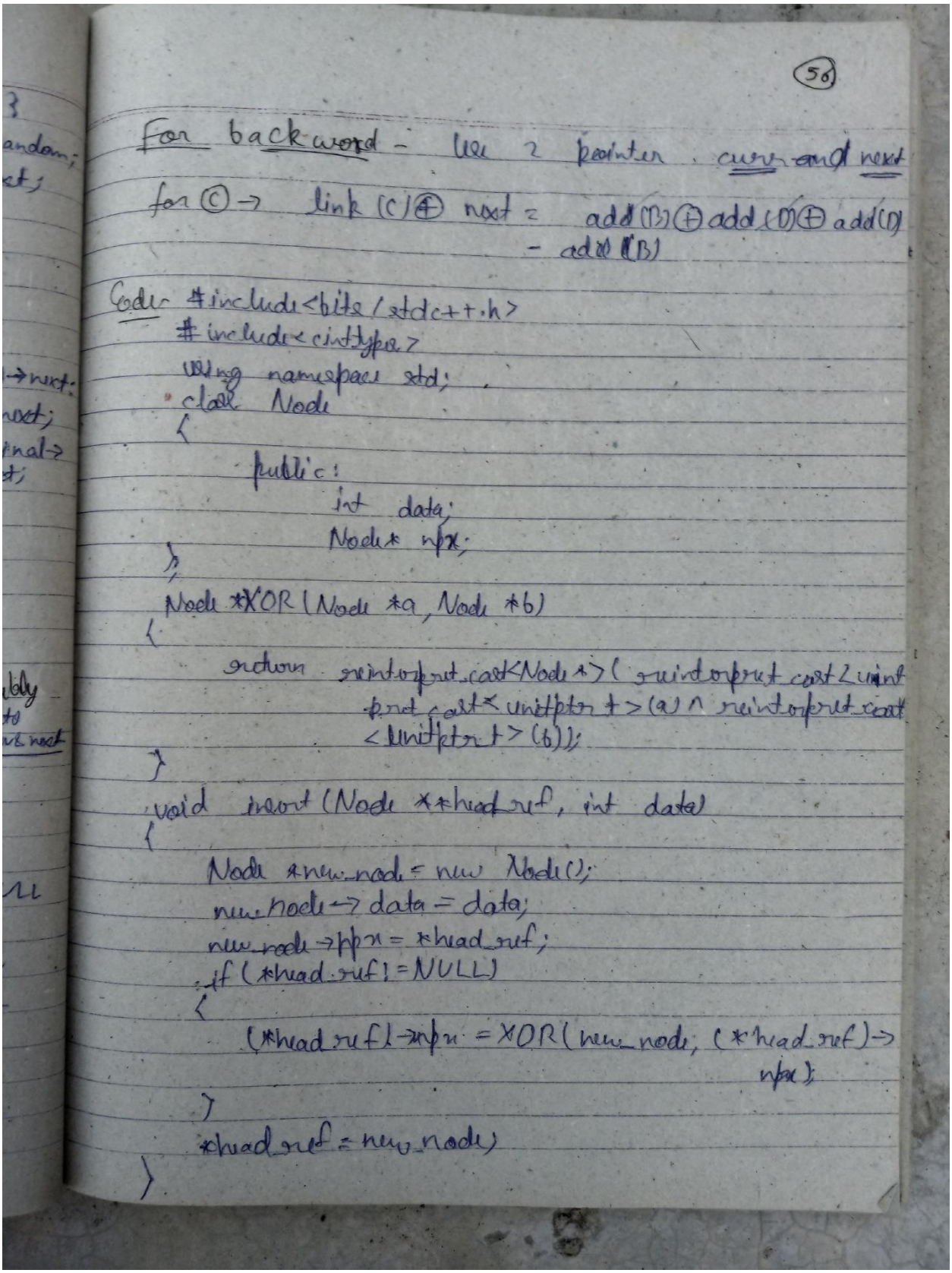


Q - How do we traverse?

For Forward for (A) → link(A) ⊕ prev = NULL ⊕ add(B) ⊕ NULL
= add(B)

For (B) → link(B) + prev = add(A) ⊕ add(C) ⊕ add(A)

Use two pointers → curr and prev



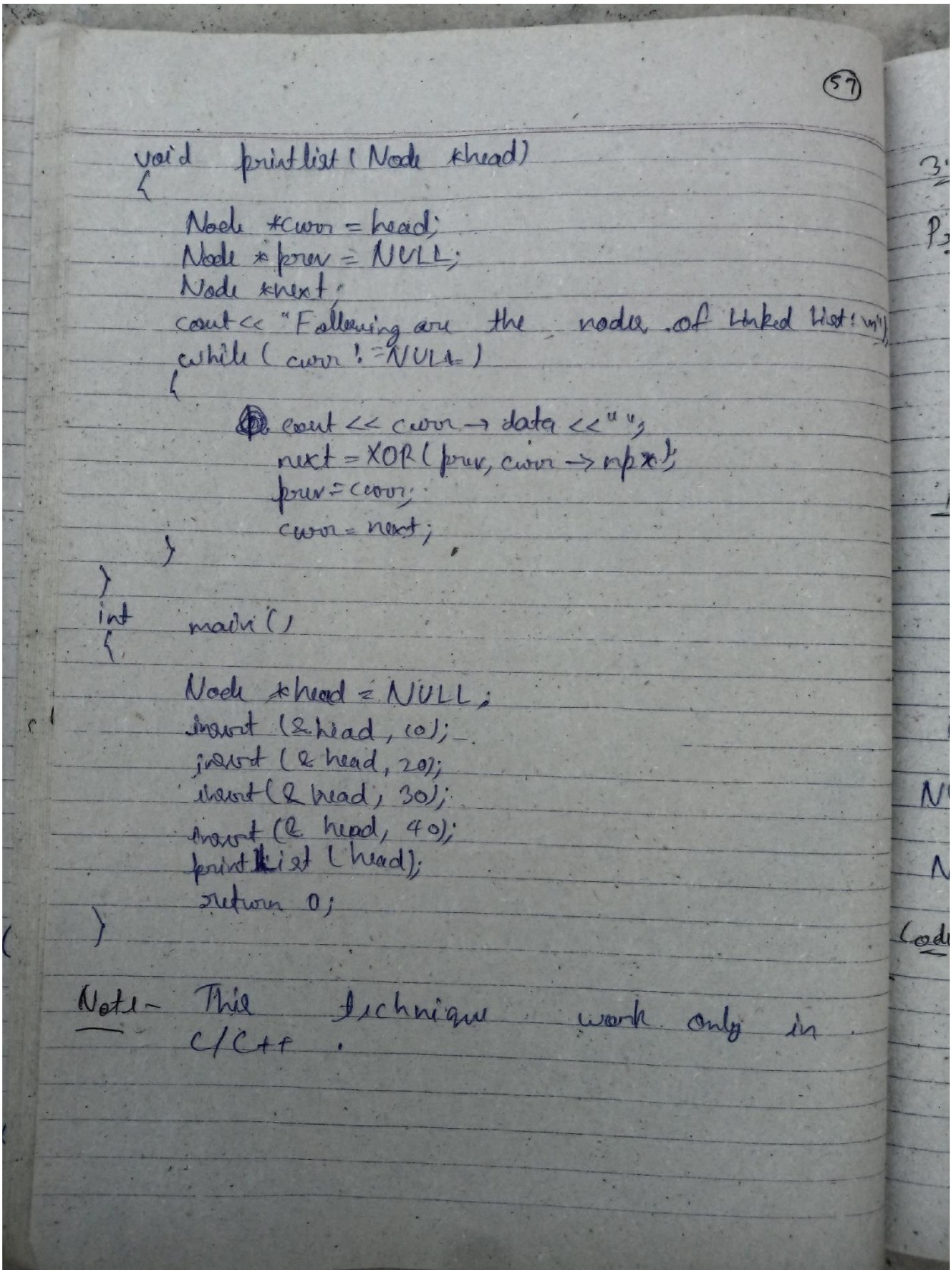
(56)

For backword - We 2 pointer curr and next

for ⓒ → link (c) ⊕ next = add(B) ⊕ add(D) ⊕ add(D) - add(B)

Code

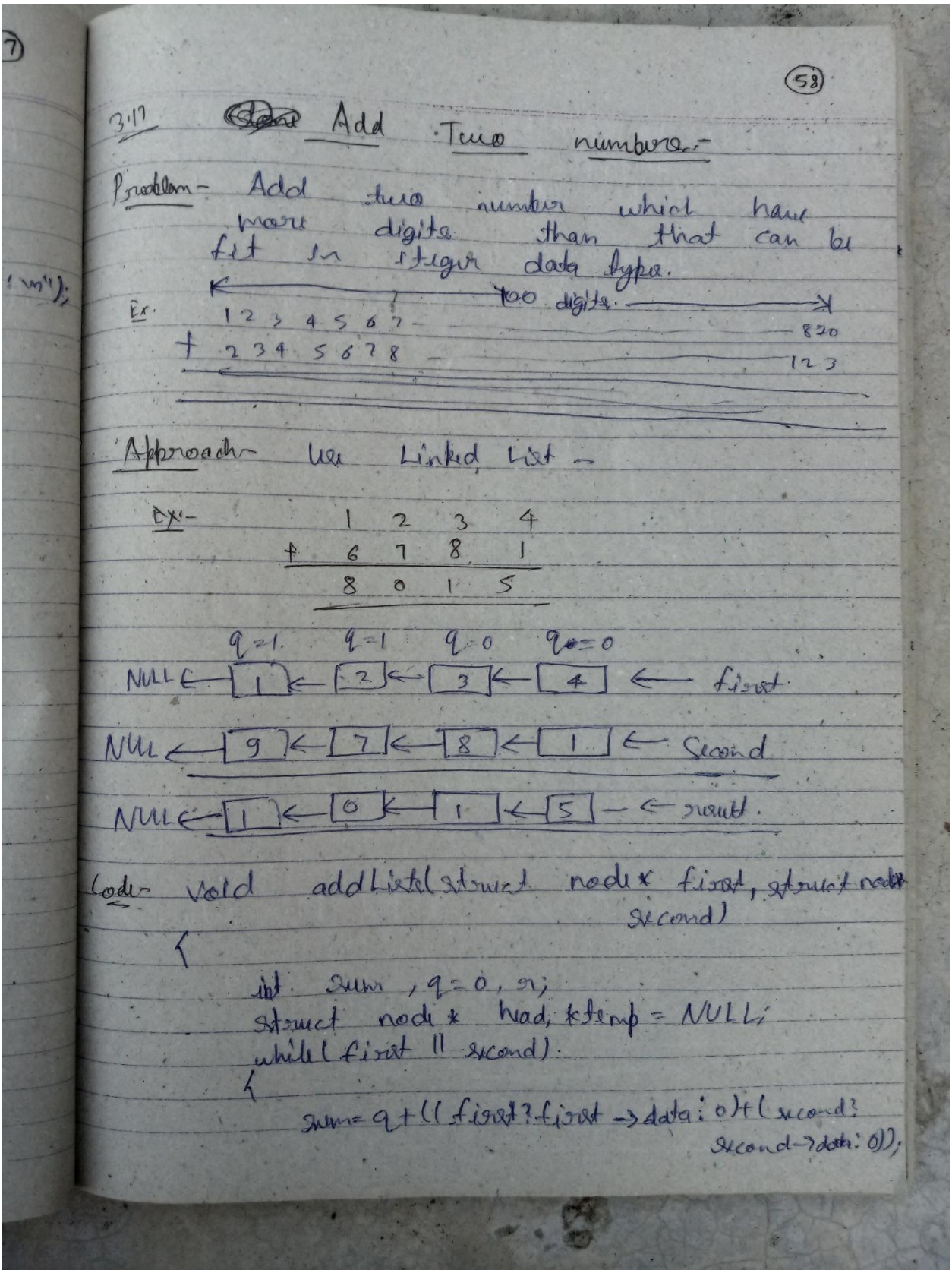
```
#include<bits/stdc++.h>
#include<cinttypes>
using namespace std;
class Node
{
    public:
        int data;
        Node* npx;
};
Node *XOR(Node *a, Node *b)
{
    return reinterpret_cast<Node*>(reinterpret_cast<uintptr_t>(a) ^ reinterpret_cast<uintptr_t>(b));
}
void insert(Node **head_ref, int data)
{
    Node *new_node = new Node();
    new_node->data = data;
    new_node->npx = *head_ref;
    if (*head_ref != NULL)
    {
        (*head_ref)->npx = XOR(new_node, (*head_ref)->npx);
    }
    *head_ref = new_node;
}
```



(57)

```
void printlist (Node *head)
{
    Node *curr = head;
    Node *prev = NULL;
    Node *next;
    cout << "Following are the nodes of Linked list: \n";
    while (curr != NULL)
    {
        cout << curr -> data << " ";
        next = XOR (prev, curr -> npx);
        prev = curr;
        curr = next;
    }
}
int main()
{
    Node *head = NULL;
    insert (&head, 10);
    insert (&head, 20);
    insert (&head, 30);
    insert (&head, 40);
    printlist (head);
    return 0;
}
```

Note- This technique work only in C/C++.



(58)

3.17 Add Two numbers :-

Problem - Add two number which have more digits than that can be fit in integer data type.

Ex. 1 2 3 4 5 6 7 - 8 2 0
+ 2 3 4 5 6 7 8 - 1 2 3

← too digits →

Approach - Use Linked List -

Ex:-
```
  1 2 3 4
+ 6 7 8 1
-------
  8 0 1 5
```

q=1 q=1 q=0 q=0

NULL ← [1] ← [2] ← [3] ← [4] ← first

NULL ← [9] ← [7] ← [8] ← [1] ← second

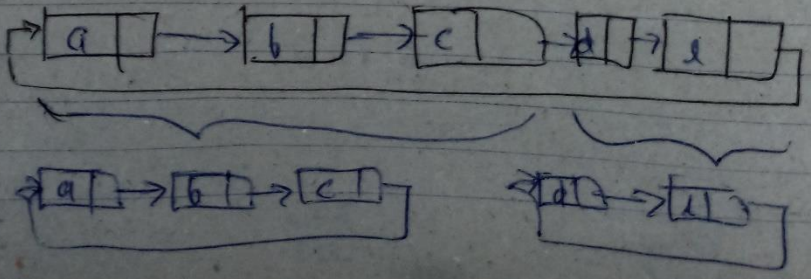
NULL ← [1] ← [0] ← [1] ← [5] ← result.

Code- void addList(struct node * first, struct node * second)

```
{
    int sum, q=0, r;
    struct node * head, *temp = NULL;
    while(first || second)
    {
        sum = q + ((first?first->data:0) + (second?second->data:0));
```

```
r = sum % 10;
q = sum / 10;
if (temp = = NULL)
{
    temp = newNode(r);
    head = temp;
}
else
{
    temp -> next = newNode(r);
    temp = temp -> next;
}
if (first)
    first = first -> next;
if (second)
    second = second -> next;
}
if (q != 0)
{
    temp -> next = newNode(r);
    temp = temp -> next;
}
}
```

3.19 Split a Circular Linked List into two halves -

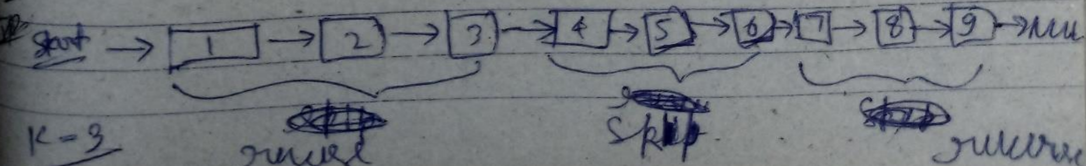


Approach- Use fast and Slow Pointers. Time: O(n), Space = O(1)

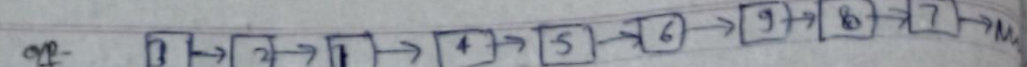
Code-

```
void splitList(struct node *head, struct node **head1, struct node **head2)
{
    struct node *slowptr = head;
    struct node *fastptr = head;
    if (head == NULL)
    {
        printf("Memory Error");
        return;
    }
    while (fastptr->next != head && fastptr->next->next != head)
    {
        fastptr = fastptr->next->next;
        slowptr = slowptr->next;
    }
    if (fastptr->next->next == head)
        fastptr = fastptr->next;
    *head1 = head;
    if (head->next != head)
        *head2 = slowptr->next;
    fastptr->next = slowptr->next;
    slow->next = head;
}
```

3.20 Reverse K alternative nodes in a linked list

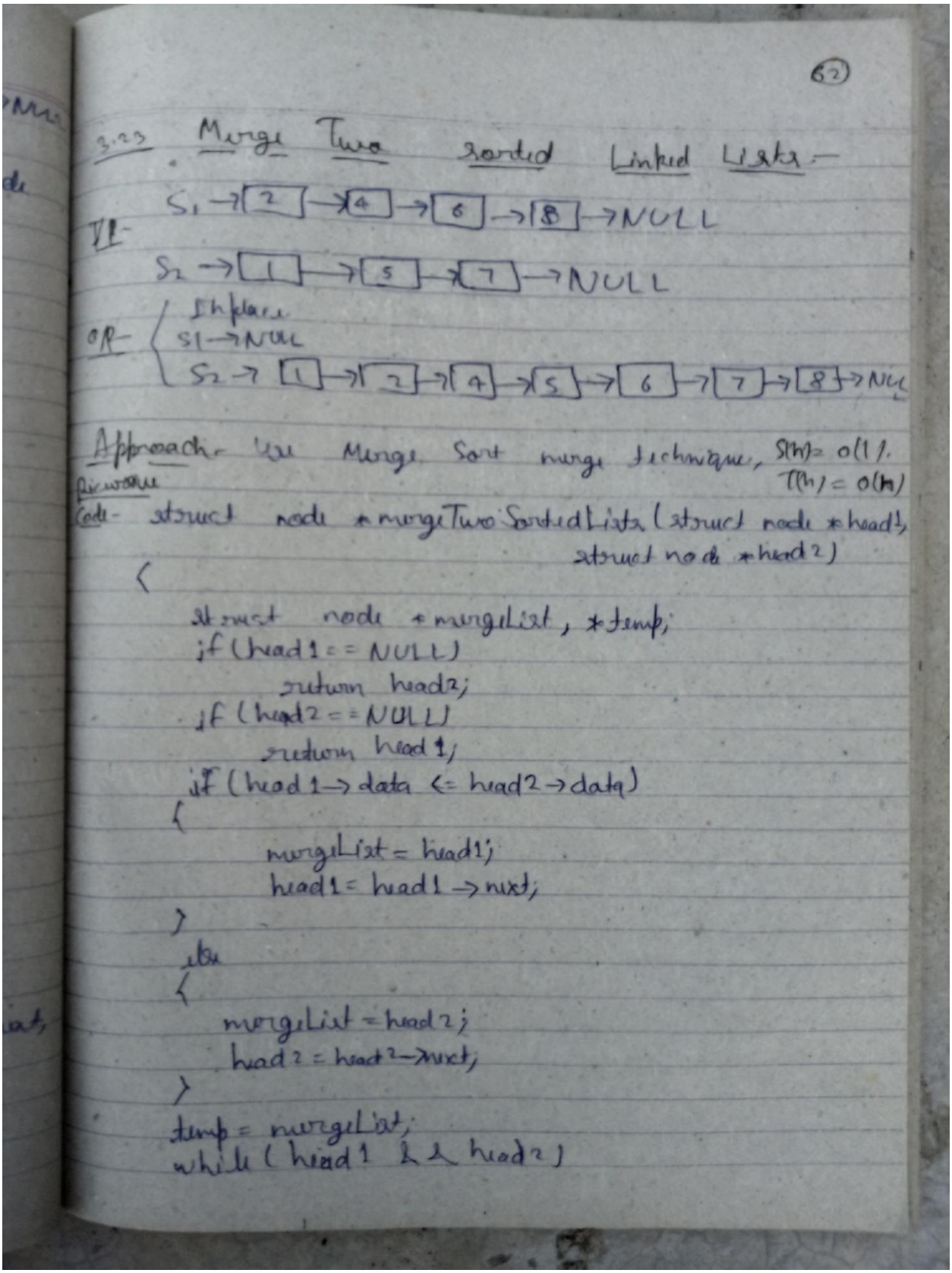


(61)



Code-

```
struct node *kAlternateReverse(struct node
                               *head, int k)
{
    struct node * curr = head;
    struct node * next;
    struct node *prev = NULL;
    int count = 0;
    while (curr != NULL && count < k)
    {
        next = curr -> next;
        curr -> next = prev;
        prev = curr;
        curr = next;
        count ++;
    }
    if (head != NULL)
        head -> next = curr;
    count = 0;
    while ( count < k-1 && curr != NULL )
    {
        curr = curr -> next;
        count ++;
    }
    if (curr != NULL)
        curr -> next = kAlternateReverse(curr->next,
                                         k);
    return prev;
}
```



62

3.23 Merge Two sorted Linked Lists:-

I/P- $S_1 \rightarrow [2] \rightarrow [4] \rightarrow [6] \rightarrow [8] \rightarrow NULL$

$S_2 \rightarrow [1] \rightarrow [5] \rightarrow [7] \rightarrow NULL$

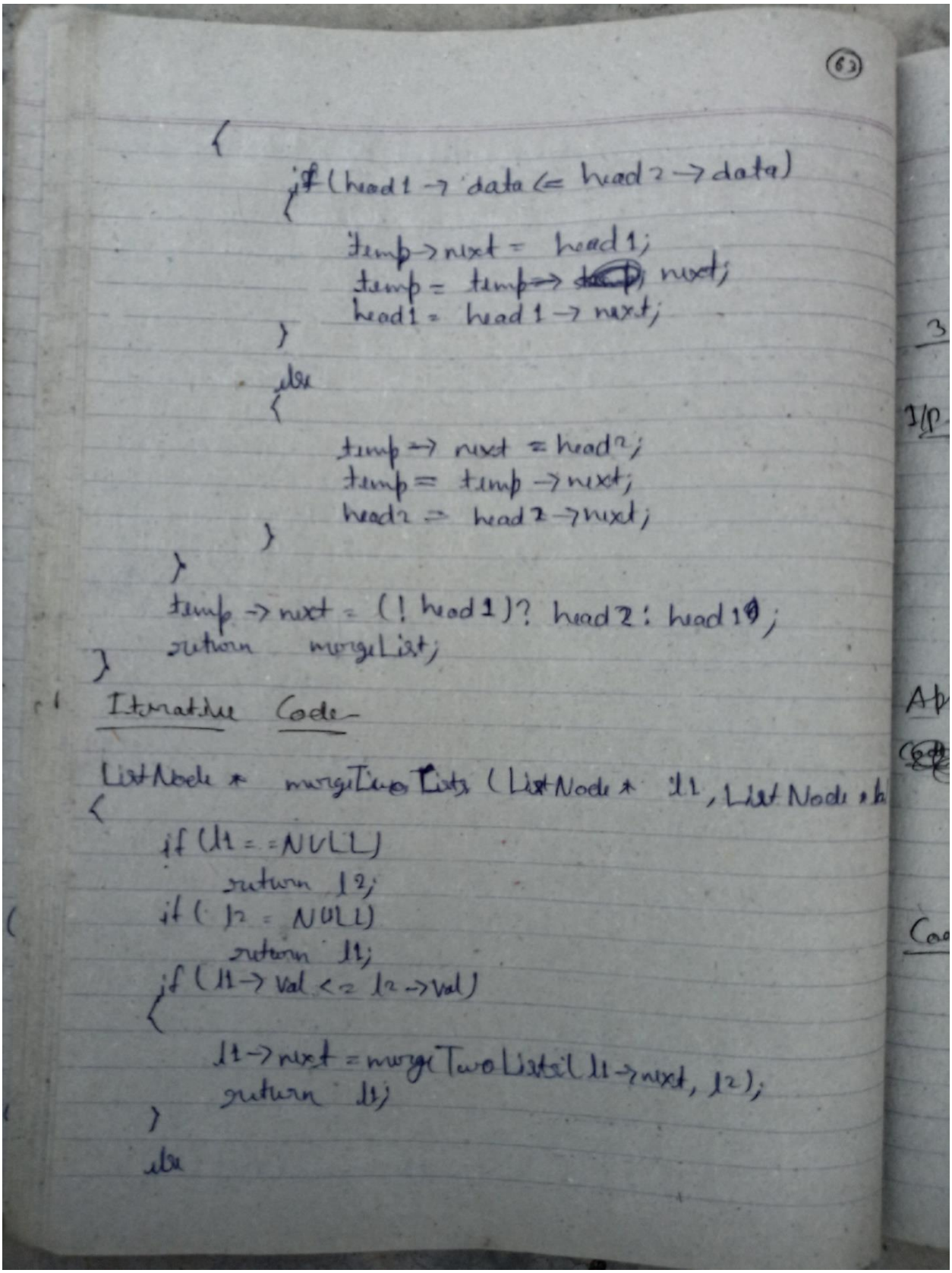
O/P- In place
$S1 \rightarrow NULL$
$S_2 \rightarrow [1] \rightarrow [2] \rightarrow [4] \rightarrow [5] \rightarrow [6] \rightarrow [7] \rightarrow [8] \rightarrow NULL$

Approach- Use Merge Sort merge technique, $S(n) = o(1)$, $T(n) = O(n)$

Pic wise

Code-

```
struct node *mergeTwoSortedLists(struct node *head1,
                                 struct node *head2)
{
    struct node *mergeList, *temp;
    if (head1 == NULL)
        return head2;
    if (head2 == NULL)
        return head1;
    if (head1->data <= head2->data)
    {
        mergeList = head1;
        head1 = head1->next;
    }
    else
    {
        mergeList = head2;
        head2 = head2->next;
    }
    temp = mergeList;
    while (head1 && head2)
```



```
    {
        if (head1 -> data <= head2 -> data)
        {
            temp -> next = head1;
            temp = temp -> next;
            head1 = head1 -> next;
        }
        else
        {
            temp -> next = head2;
            temp = temp -> next;
            head2 = head2 -> next;
        }
    }
    temp -> next = (! head1)? head2 : head1;
    return mergeList;
}
```

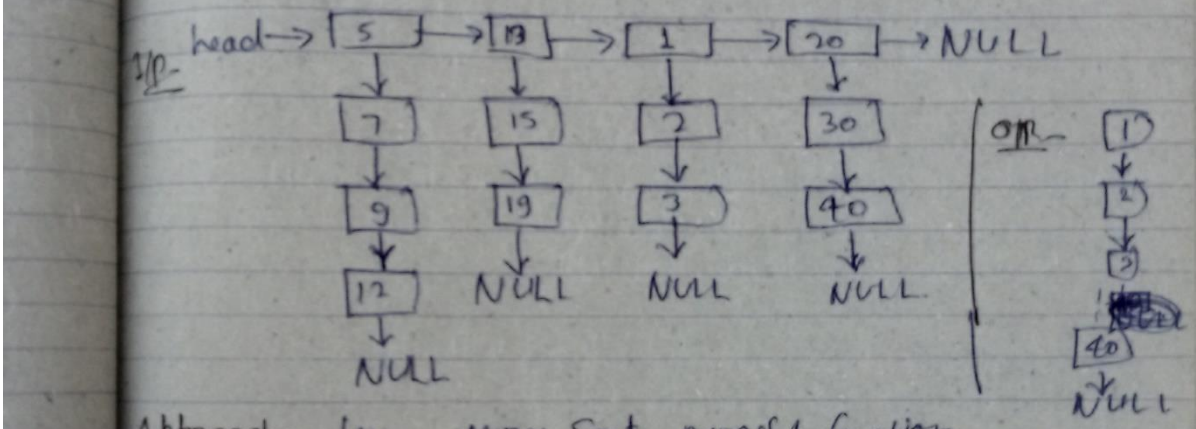
Iterative Code-

```
ListNode * mergeTwoLists (ListNode * l1, ListNode * l
{
    if (l1 == NULL)
        return l2;
    if (l2 == NULL)
        return l1;
    if (l1 -> val <= l2 -> val)
    {
        l1 -> next = mergeTwoLists(l1 -> next, l2);
        return l1;
    }
    else
```

(64)

```
        l2->next = mergeTwoLists(l2->next, l1);
        return l2;
    }
}
```

3.26 Flattening a Linked List -

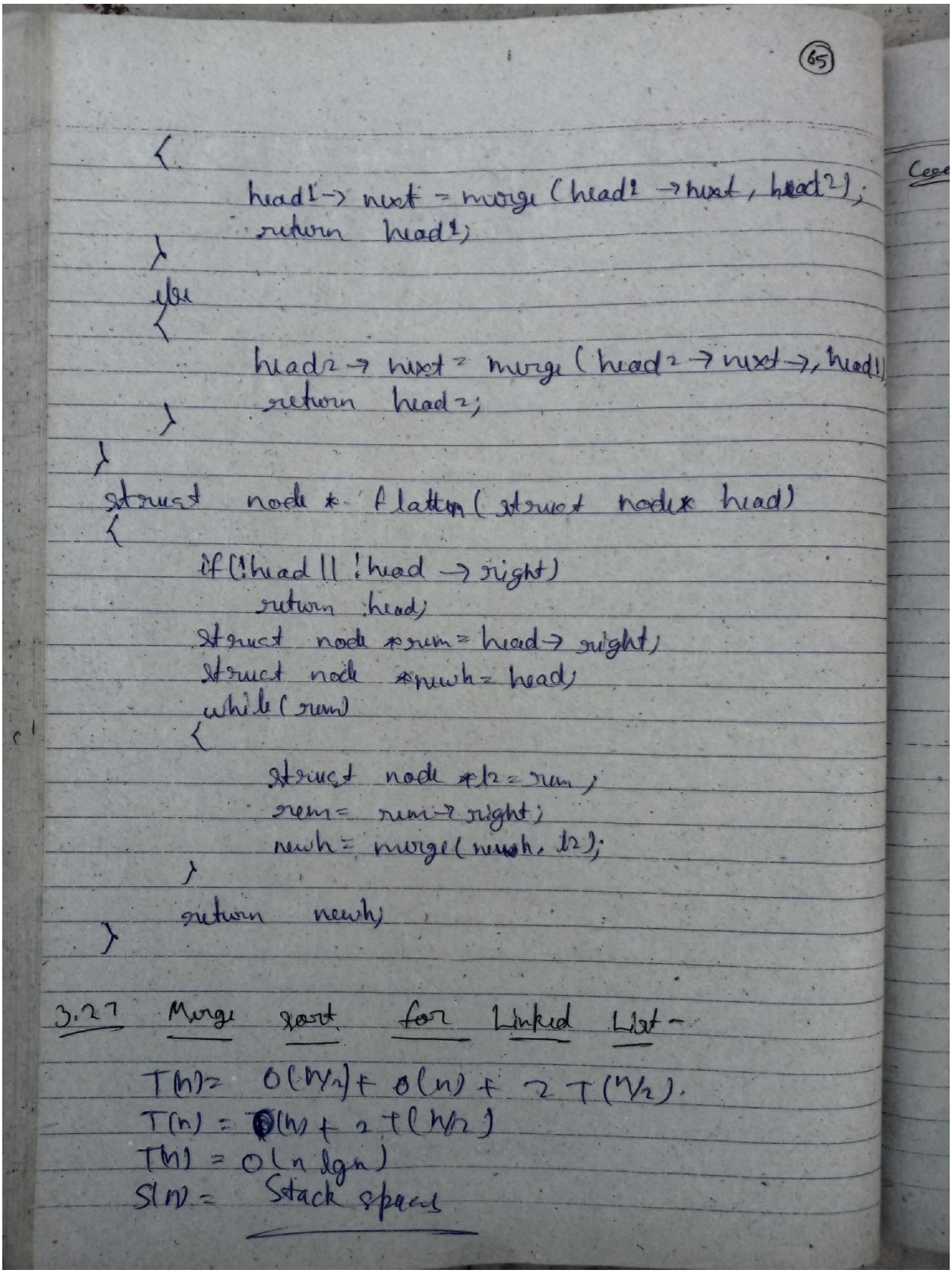


Approach- Use Merge Sort merge() function -

$$T(n) \begin{cases} O(n_1 + n_2 + n_3 \text{ ---- } n_k) \text{ if } K << \min(n_1, n_2, \ldots n_k) \\ O(n_1+n_2) + (n_1+n_2+n_3) + (n_1+n_2+n_3+n_4) \ldots \\ \qquad (n_1 + n_2 + \ldots + n_K) \end{cases}$$

Code-

```
struct node* merge(struct node* head1, struct
                                node* head2)
{
    if (!head1)
        return head2;
    if (!head2)
        return head1;
    if (head1->val <= head2->val)
```



```
    {
        head1 -> next = merge(head1 -> next, head2);
        return head1;
    }
    else
    {
        head2 -> next = merge(head2 -> next, head1);
        return head2;
    }
}
struct node * flatten(struct node* head)
{
    if(!head || !head -> right)
        return head;
    struct node *rem = head -> right;
    struct node *newh = head;
    while(rem)
    {
        struct node *l2 = rem;
        rem = rem -> right;
        newh = merge(newh, l2);
    }
    return newh;
}
```

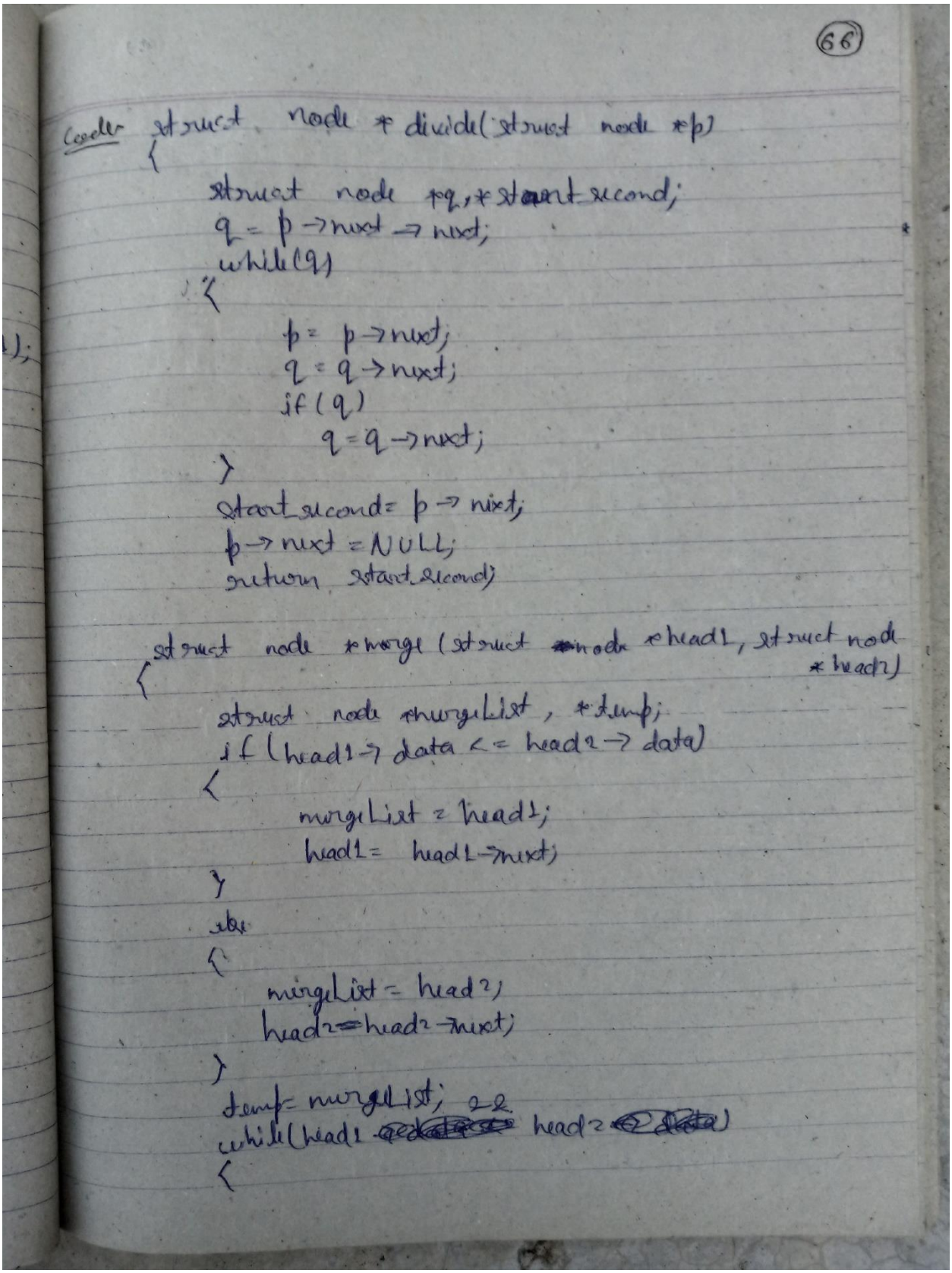
3.27 Merge sort for Linked List -

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

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

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

$S(n) =$ Stack space

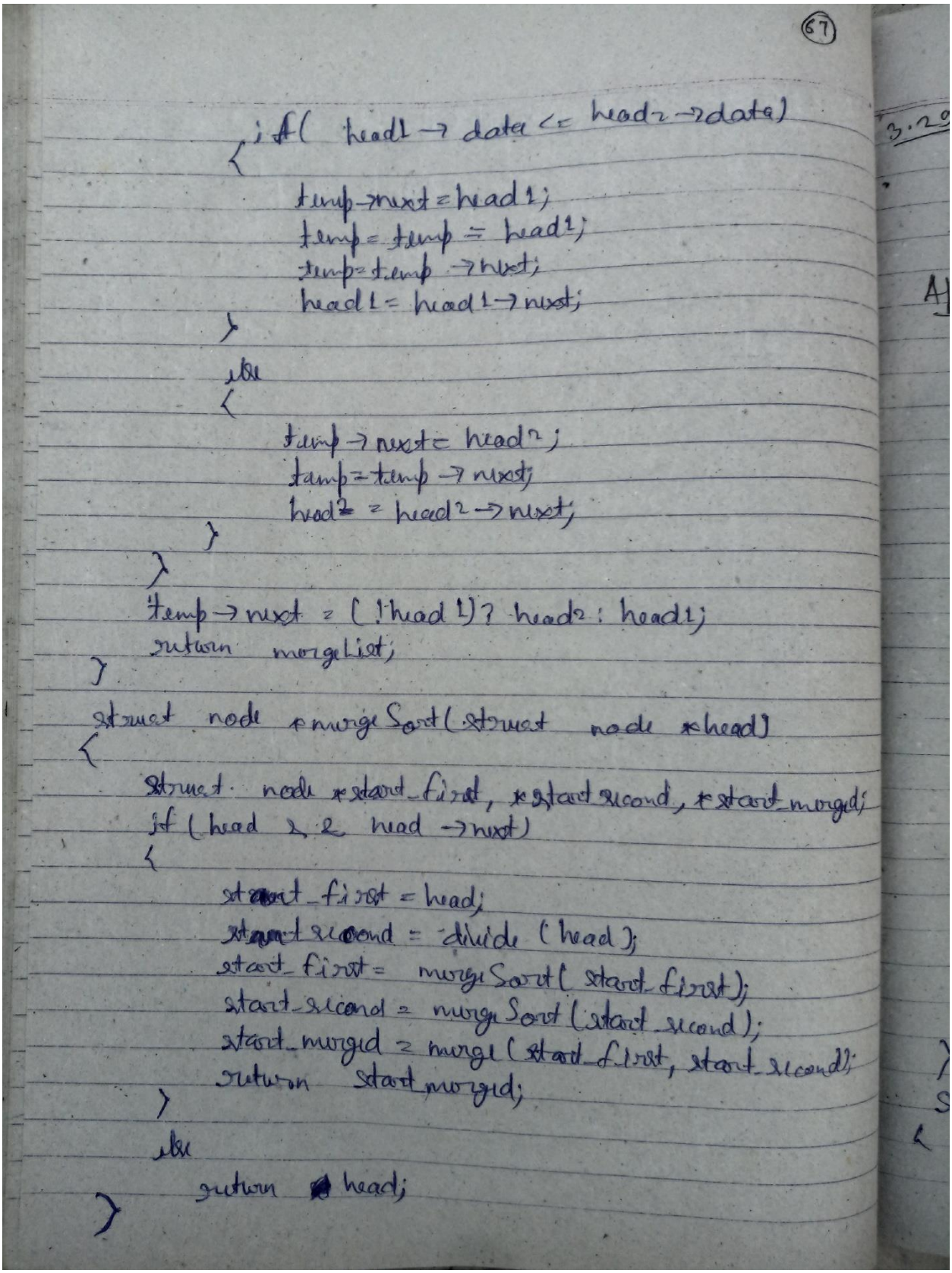


(66)

Coder

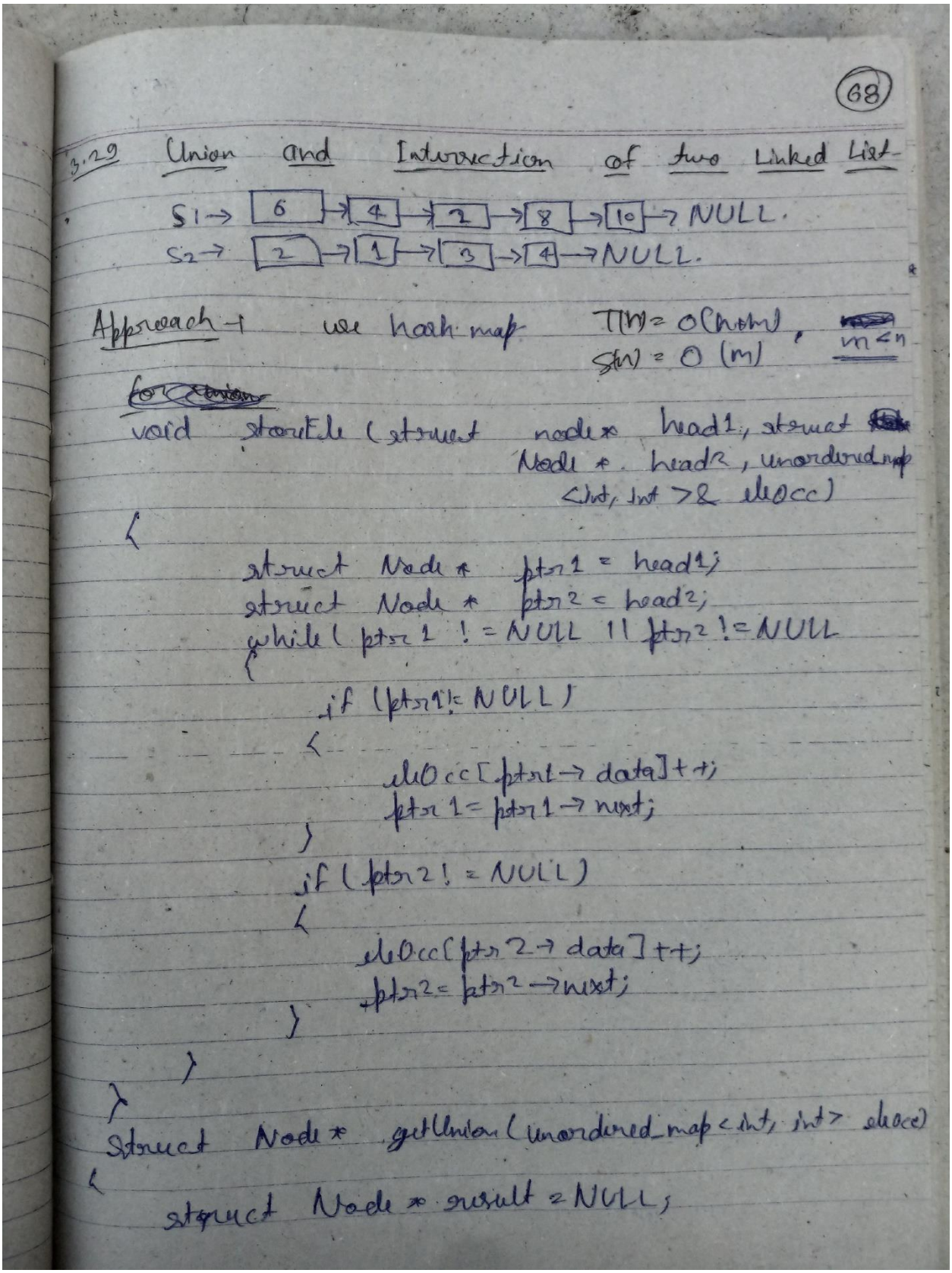
```
struct node * divide(struct node *p)
{
    struct node *q, *start_second;
    q = p->next->next;
    while(q)
    {
        p = p->next;
        q = q->next;
        if(q)
            q = q->next;
    }
    start_second = p->next;
    p->next = NULL;
    return start_second;
}

struct node *merge(struct node *head1, struct node *head2)
{
    struct node *mergeList, *temp;
    if(head1->data <= head2->data)
    {
        mergeList = head1;
        head1 = head1->next;
    }
    else
    {
        mergeList = head2;
        head2 = head2->next;
    }
    temp = mergeList;
    while(head1 && head2)
    {
```



```
        if( head1 -> data <= head2 -> data)
        {
            temp->next = head1;
            temp = temp = head1;
            temp = temp -> next;
            head1 = head1 -> next;
        }
        else
        {
            temp -> next = head2;
            temp = temp -> next;
            head2 = head2 -> next;
        }
    }
    temp -> next = (!head1)? head2 : head1;
    return mergeList;
}

struct node *mergeSort(struct node *head)
{
    struct node *start_first, *start_second, *start_merged;
    if(head != NULL && head -> next)
    {
        start_first = head;
        start_second = divide(head);
        start_first = mergeSort(start_first);
        start_second = mergeSort(start_second);
        start_merged = merge(start_first, start_second);
        return start_merged;
    }
    else
        return head;
}
```



68

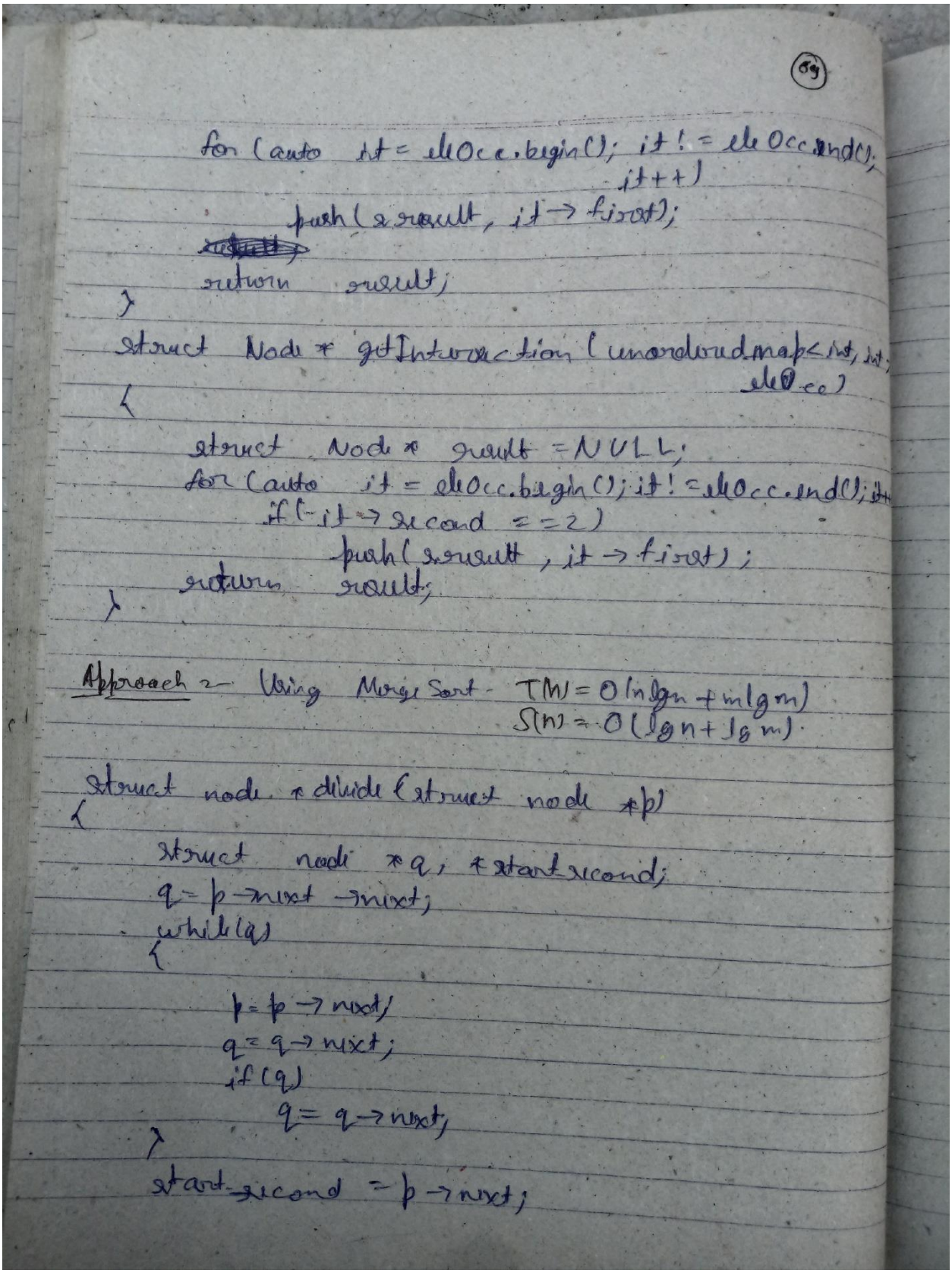
3.29 Union and Intersection of two Linked List

S1 → 6 → 4 → 2 → 8 → 10 → NULL.

S2 → 2 → 1 → 3 → 4 → NULL.

Approach → use hash map T(n) = O(n+m), S(n) = O(m), m<n

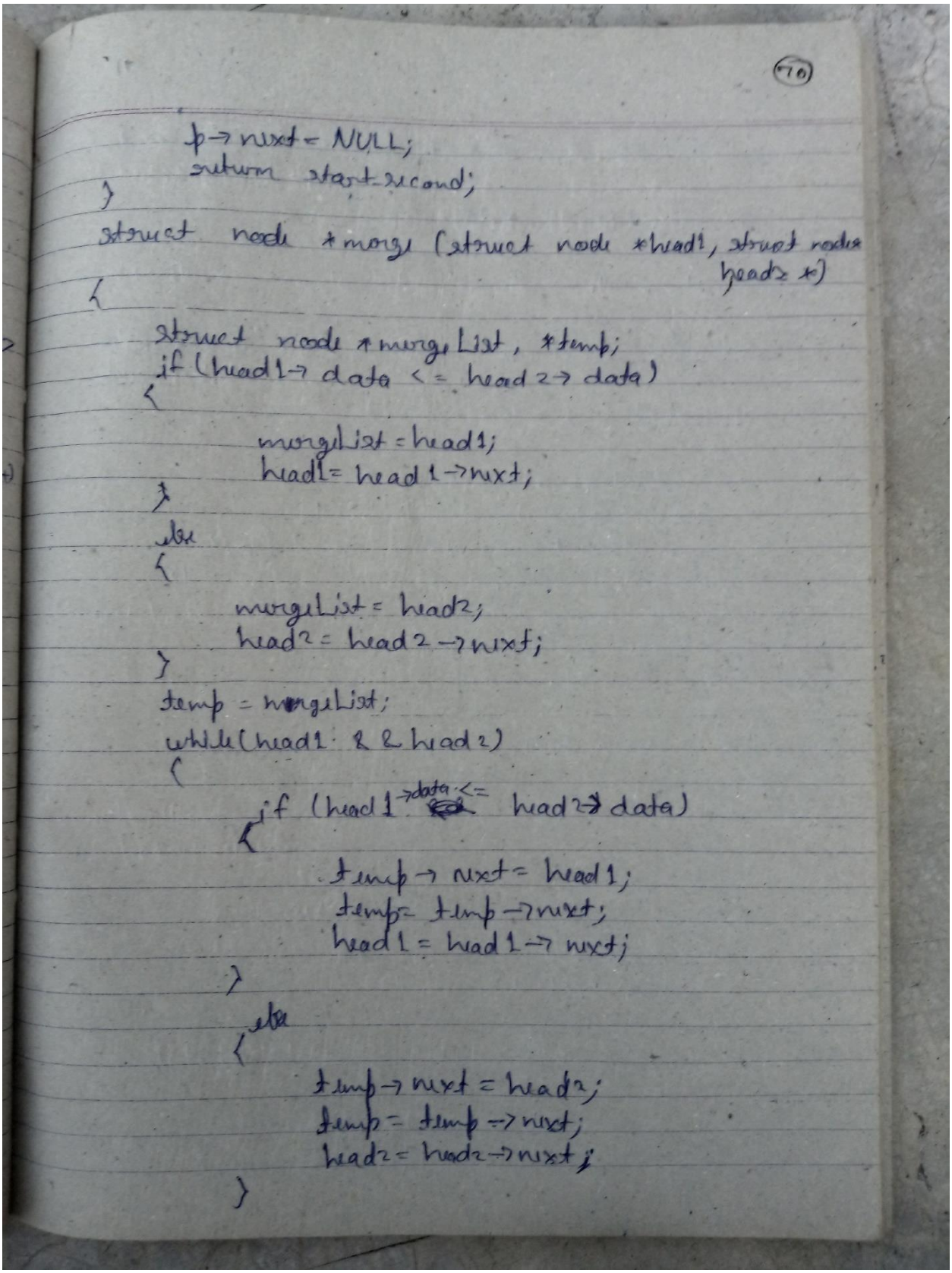
```
void storeEle (struct node* head1, struct Node * head2, unordered_map <int, int>& eleOcc)
{
    struct Node * ptr1 = head1;
    struct Node * ptr2 = head2;
    while ( ptr1 != NULL || ptr2 != NULL
    {
        if (ptr1 != NULL)
        {
            eleOcc[ptr1->data]++;
            ptr1 = ptr1->next;
        }
        if (ptr2 != NULL)
        {
            eleOcc[ptr2->data]++;
            ptr2 = ptr2->next;
        }
    }
}
Struct Node * getUnion (unordered_map<int, int> eleOcc)
{
    struct Node * result = NULL;
```



```
for (auto it = elOcc.begin(); it != elOcc.end();
                                    it++)
        push(result, it->first);
    return result;
}
struct Node * getIntersection (unordered_map<int, int>
                                    elO_cc)
{
    struct Node * result = NULL;
    for (auto it = elOcc.begin(); it != elOcc.end(); it
        if (it->second == 2)
            push(result, it->first);
    return result;
}
```

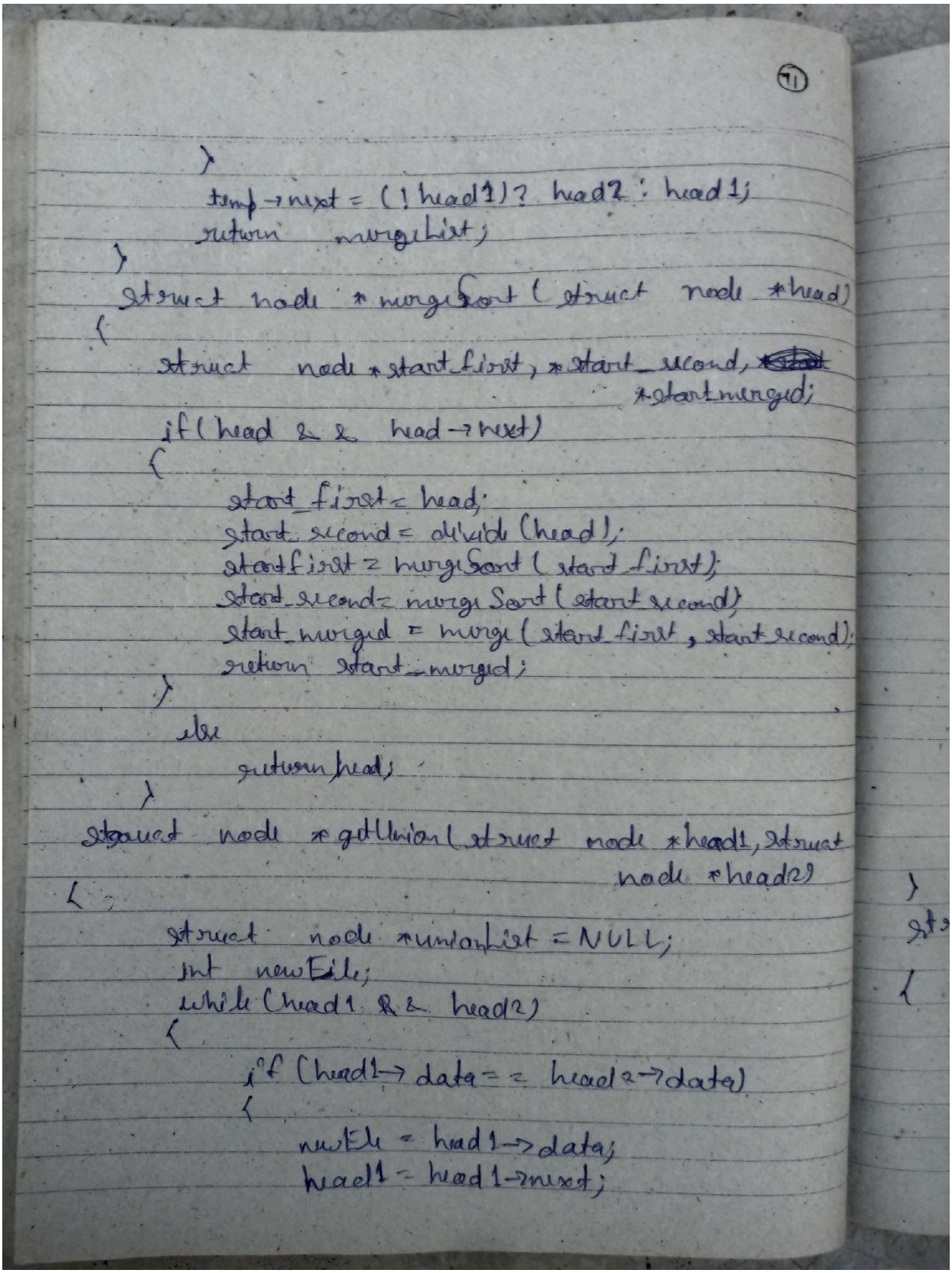
Approach :- Using Merge Sort - $T(n) = O(n \lg n + m \lg m)$
$S(n) = O(\lg n + \lg m)$.

```
struct node * divide (struct node *p)
{
    struct node *q, *start_second;
    q = p->next->next;
    while(q)
    {
        p = p->next;
        q = q->next;
        if(q)
            q = q->next;
    }
    start_second = p->next;
```

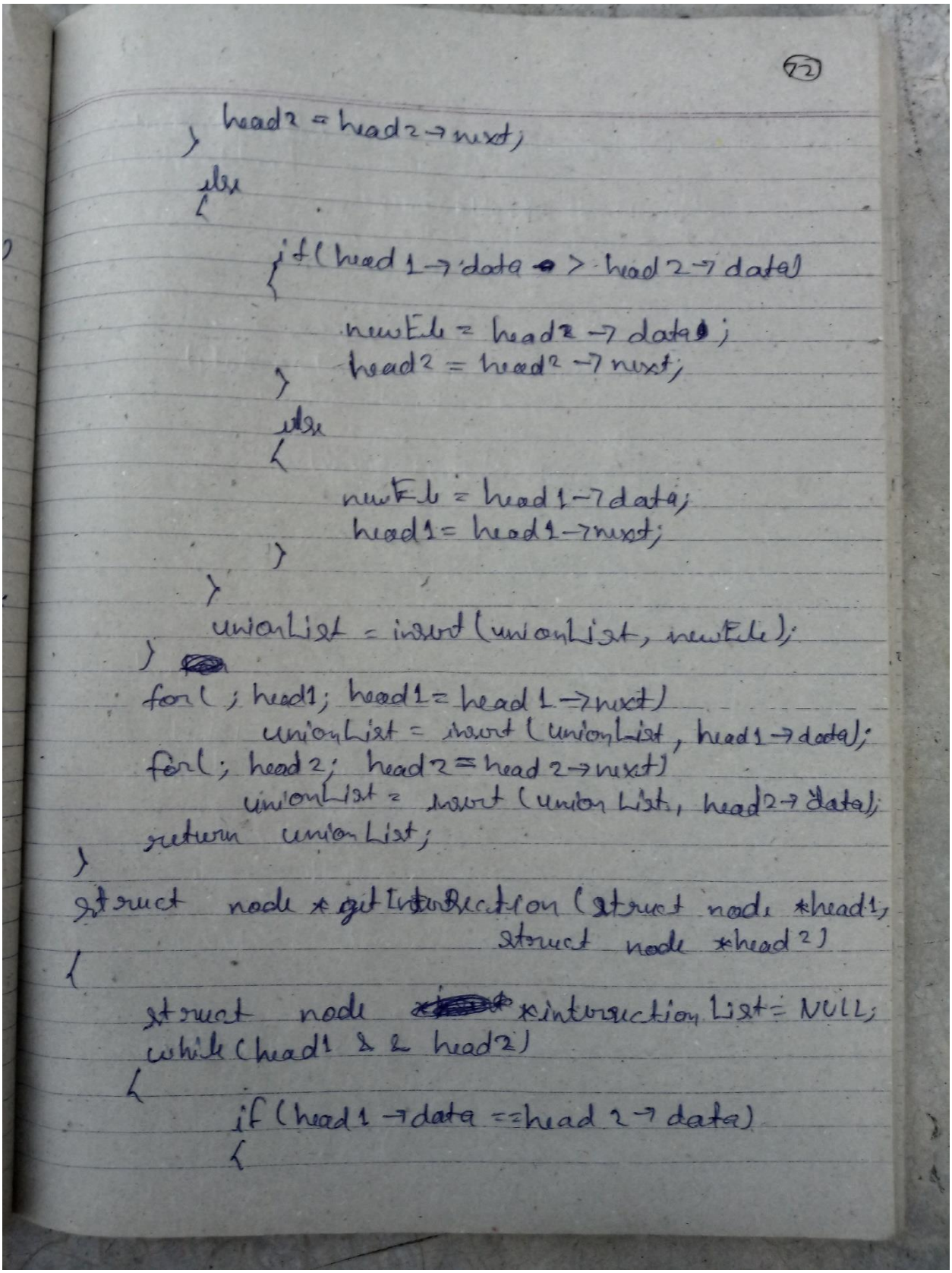


```
(70)

    p->next = NULL;
    return start->second;
}
struct node *merge (struct node *head1, struct node
                                        head2 *)
{
    struct node *mergeList, *temp;
    if (head1->data <= head2->data)
    {
        mergeList = head1;
        head1 = head1->next;
    }
    else
    {
        mergeList = head2;
        head2 = head2->next;
    }
    temp = mergeList;
    while(head1 && head2)
    {
        if (head1->data <= head2->data)
        {
            temp->next = head1;
            temp = temp->next;
            head1 = head1->next;
        }
        else
        {
            temp->next = head2;
            temp = temp->next;
            head2 = head2->next;
        }
```



```
        }
        temp->next = (!head1)? head2 : head1;
        return mergelist;
    }
    struct node *mergeSort (struct node *head)
    {
        struct node *start_first, *start_second,
                    *startmerged;
        if(head && head->next)
        {
            start_first = head;
            start_second = divide(head);
            startfirst = mergeSort(start_first);
            start_second = mergeSort(start_second)
            start_merged = merge(start_first, start_second);
            return start_merged;
        }
        else
            return head;
    }
    struct node *getUnion(struct node *head1, struct
                          node *head2)
    {
        struct node *unionlist = NULL;
        int newEle;
        while(head1 && head2)
        {
            if(head1->data == head2->data)
            {
                newEle = head1->data;
                head1 = head1->next;
```



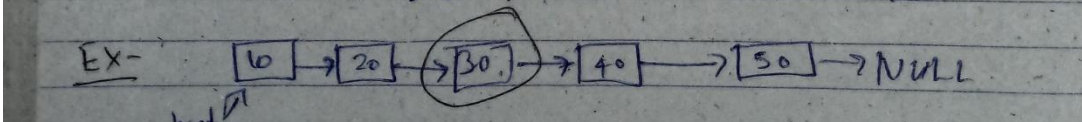
```
                                                    (72)
        head2 = head2->next;
    }
    else
    {
        if(head1->data > head2->data)
        {
            newEle = head2->data;
            head2 = head2->next;
        }
        else
        {
            newEle = head1->data;
            head1 = head1->next;
        }
    }
    unionList = insert(unionList, newEle);
}
for( ; head1; head1 = head1->next)
    unionList = insert(unionList, head1->data);
for( ; head2; head2 = head2->next)
    unionList = insert(unionList, head2->data);
return unionList;
}
struct node *getIntersection(struct node *head1,
                             struct node *head2)
{
struct node *intersectionList = NULL;
while(head1 && head2)
{
    if(head1->data == head2->data)
    {
```

```
        intersectionList = insert(intersectionList, head1->data);
        head1 = head1->next;
        head2 = head2->next;
    }
    else
    {
        if (head1->data > head->data)
            head2 = head2->next;
        else
            head1 = head1->next;
    }
}
return intersectionList;
}
```

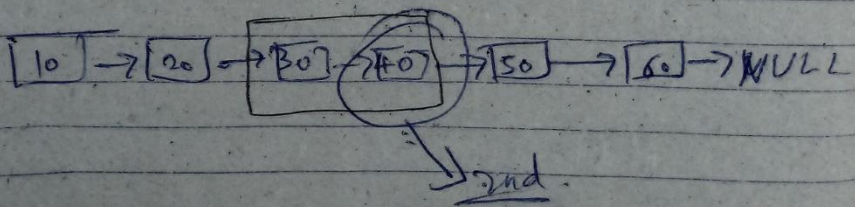
3.36 Find Middle Element in a Linked List-

Problem Statement- Given a non-empty, singly linked list with head node "head", return a middle node of linked list. If there are two middle nodes, return the second middle node.

EX-



head =>



(74)

Approach 1- Count element. T(n) = O(n) S(n) = O(1)

Code-
```
ListNode *middleNode(ListNode * head)
{
    if(head == NULL || head->next == NULL)
        return head;
    int count = 0;
    ListNode *temp = head;
    while(temp)
    {
        count++;
        temp = temp->next;
    }
    int n = count/2;
    temp = head;
    while(n--)
        temp = temp->next;
    return temp;
}
```

Approach 2- Use 2 pointer Fast & Slow. T(n) = O(n) S(n) = O(1)

Code-
```
ListNode *middleNode(ListNode * head)
{
    ListNode *fast = head, *slow = head;
    while(fast && fast->next)
    {
        fast = fast->next->next;
        slow = slow->next;
    }
    return slow;
}
```

3.37 Reverse Linked List -

Code -

```
ListNode *reverseList(ListNode *head)
{
    ListNode *prev = NULL;
    ListNode *current = head;
    ListNode *next = NULL;
    while(current != NULL)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
```

3.38 Remove Duplicates from Sorted linked list

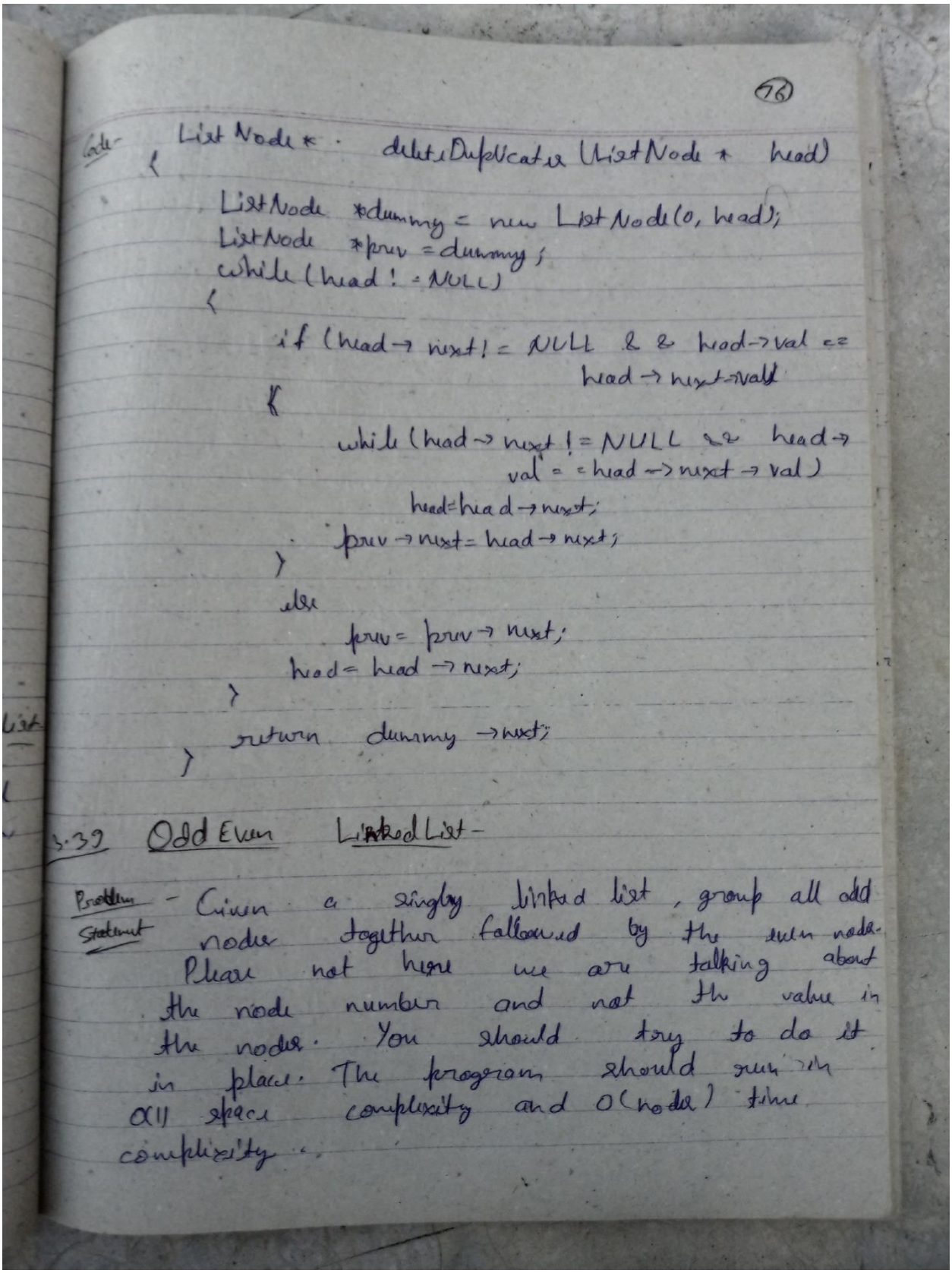
Problem Statement - Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

Ex- I/P - 1 → 2 → 3 → 3 → 4 → 5.

O/P - 1 → 2 → 5

Ex- I/P - 1 → 1 → 1 → 2 → 3

O/P - 2 → 3

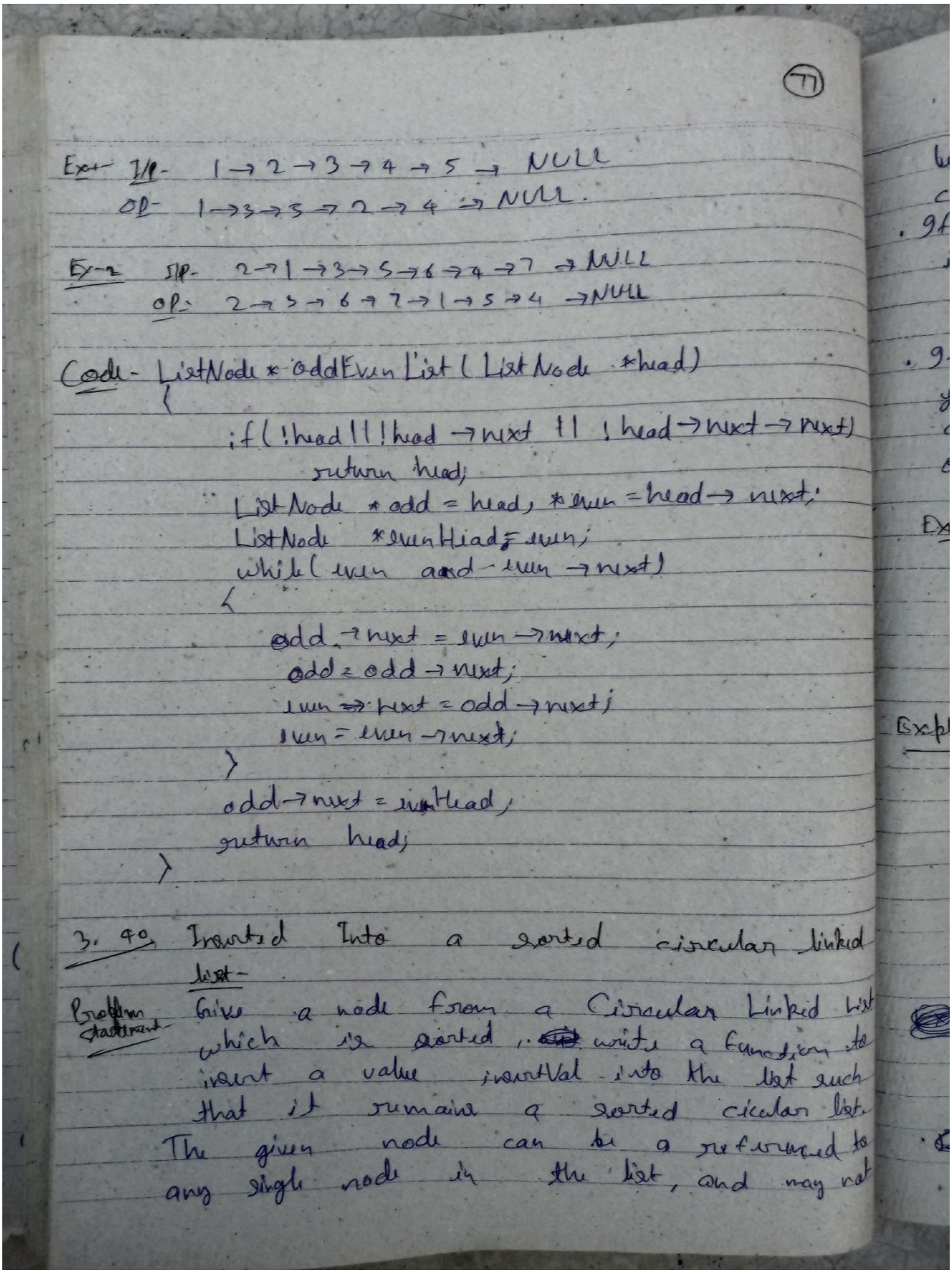


76

```
Code-  ListNode* deleteDuplicates(ListNode* head)
{
    ListNode *dummy = new ListNode(0, head);
    ListNode *prev = dummy;
    while (head != NULL)
    {
        if (head->next != NULL && head->val ==
                                    head->next->val)
        {
            while (head->next != NULL && head->
                                    val == head->next->val)
                head=head->next;
            prev->next= head->next;
        }
        else
            prev= prev->next;
        head= head->next;
    }
    return dummy->next;
}
```

3.39 Odd Even Linked List -

Problem Statement - Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes. You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.



(77)

Ex- I/P- 1→2→3→4→5→NULL
O/P- 1→3→5→2→4→NULL.

Ex-2 I/P- 2→1→3→5→6→4→7→NULL
O/P- 2→5→6→7→1→5→4→NULL

Code-
```
ListNode * oddEvenList ( ListNode *head)
{
    if(!head||!head->next||!head->next->next)
        return head;
    ListNode *odd = head, *even = head->next;
    ListNode *evenHead = even;
    while(even and even->next)
    {
        odd->next = even->next;
        odd = odd->next;
        even->next = odd->next;
        even = even->next;
    }
    odd->next = evenHead;
    return head;
}
```

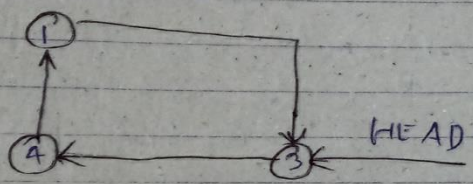
3. 4o Inserted Into a sorted circular linked list-

Problem statement- Give a node from a Circular Linked list which is sorted, write a function to insert a value insertVal into the list such that it remains a sorted circular list. The given node can be a reference to any single node in the list, and may not

(78)

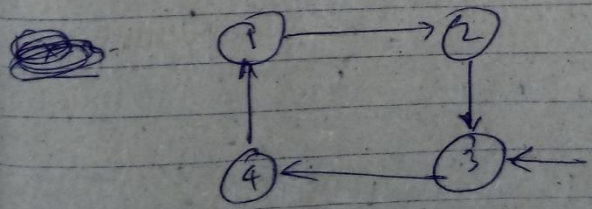
be necessarily the smallest value in the circular list.
. If there are multiple suitable places for insertion, you may choose any place to insert the new val. After the insertion, the circular list should remain sorted.
. If the list is empty (i.e, given node is null), you should create a new single circular list and return the reference to that single node. Otherwise, you should return the original given node.

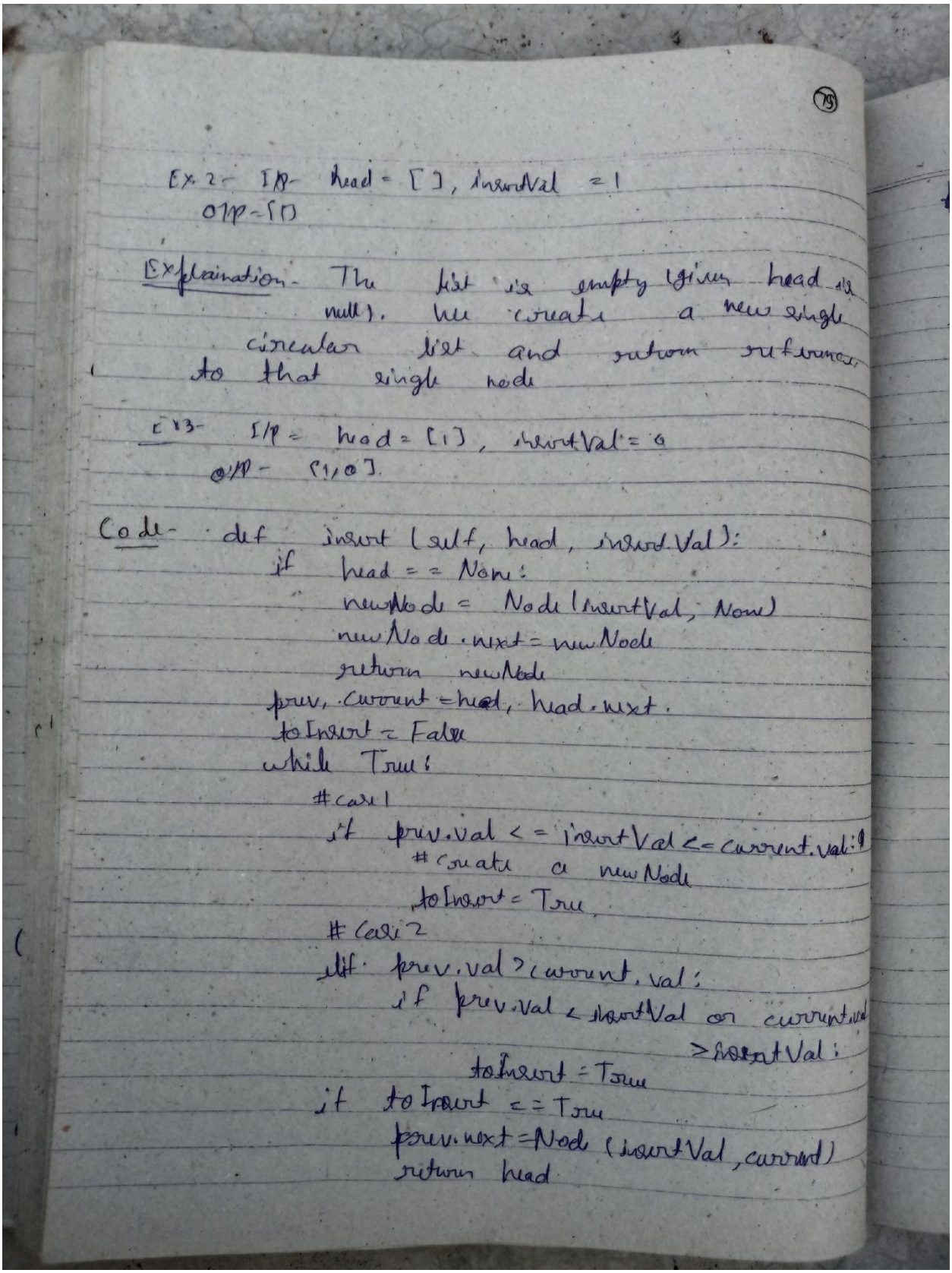
Ex1-



I/P - head = [3,4,1], insertVal = 2
O/P - [3,4,1,2]

Explanation- In the figure above, there is a sorted circular list of three elements. You are given a reference to the node with value 3, and we need to insert 2 into the list. The new node should be inserted between node 1 and node 3. After the insertion, the list should look like this, and we should still return node 3.





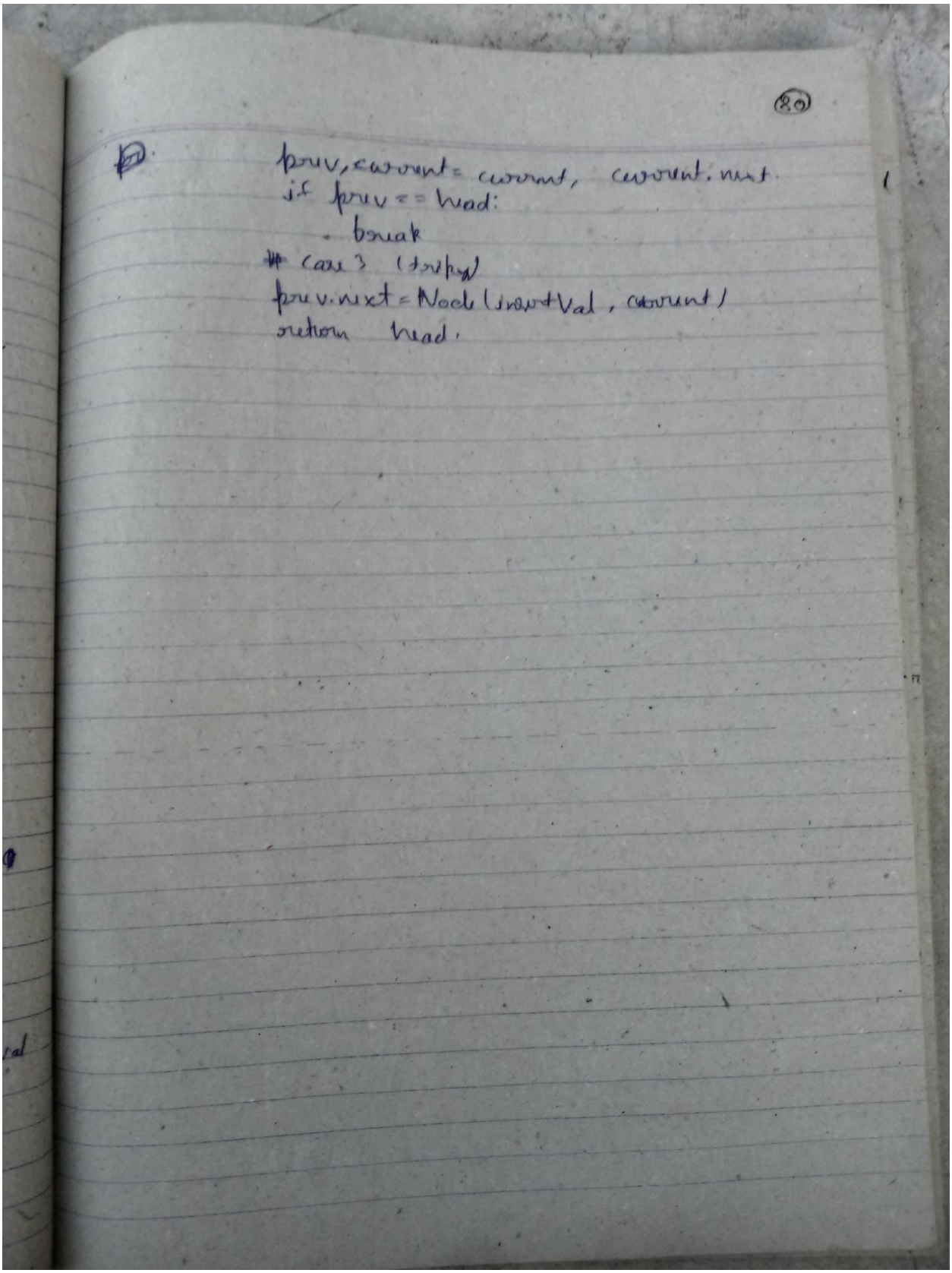
79

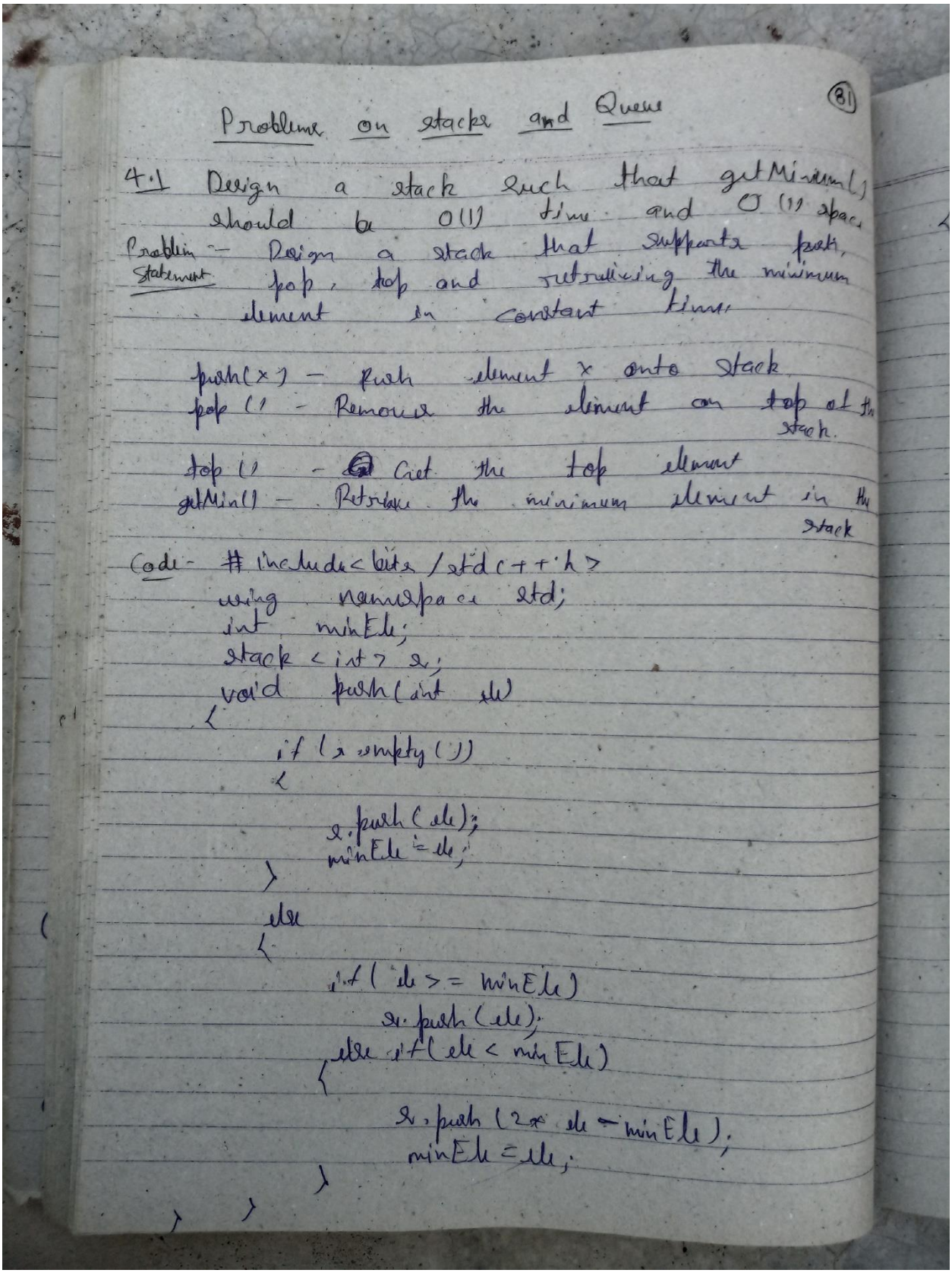
Ex 2- I/P- head = [], insertVal = 1
O/P - [1]

Explanation- The list is empty (given head is null). We create a new single circular list and return reference to that single node

Ex 3- I/P = head = [1], insertVal = 0
O/P - [1, 0].

Code-
```
def insert (self, head, insertVal):
    if head == None:
        newNode = Node(insertVal, None)
        newNode.next = newNode
        return newNode
    prev, current = head, head.next
    toInsert = False
    while True:
        #case 1
        if prev.val <= insertVal <= current.val:
            #create a new Node
            toInsert = True
        #Case 2
        elif prev.val > current.val:
            if prev.val < insertVal or current.val
                                    >= insertVal:
                toInsert = True
        if toInsert == True
            prev.next = Node(insertVal, current)
            return head
```


prev, current = current, current.next
if prev == head:
break
case 3 (tricky)
prev.next = Node(insertVal, current)
return head.



Problems on stacks and Queues

(81)

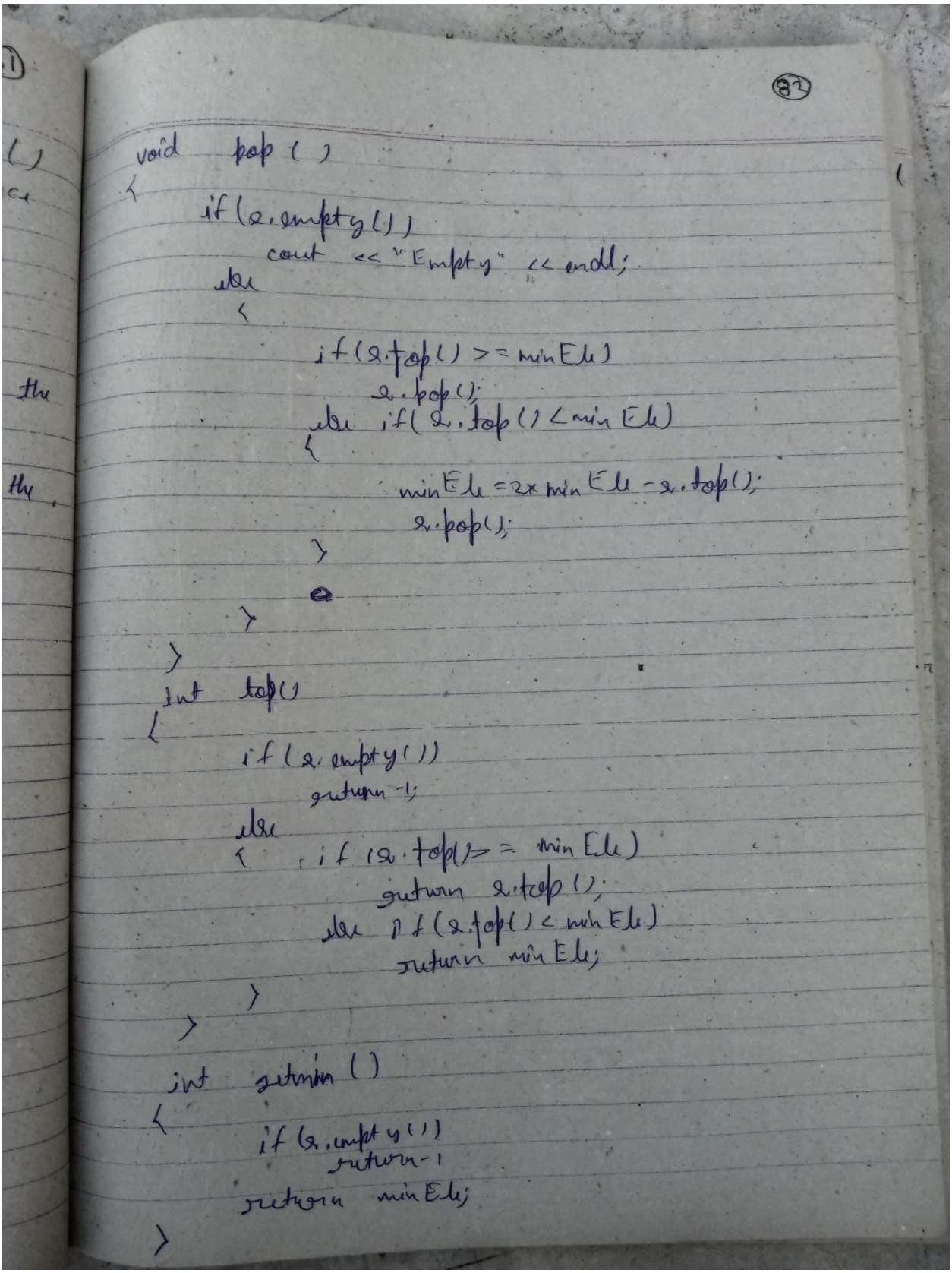
4.1 Design a stack such that getMinimum() should be O(1) time and O(1) space.

Problem Statement — Design a stack that supports push, pop, top and retrieving the minimum element in constant time.

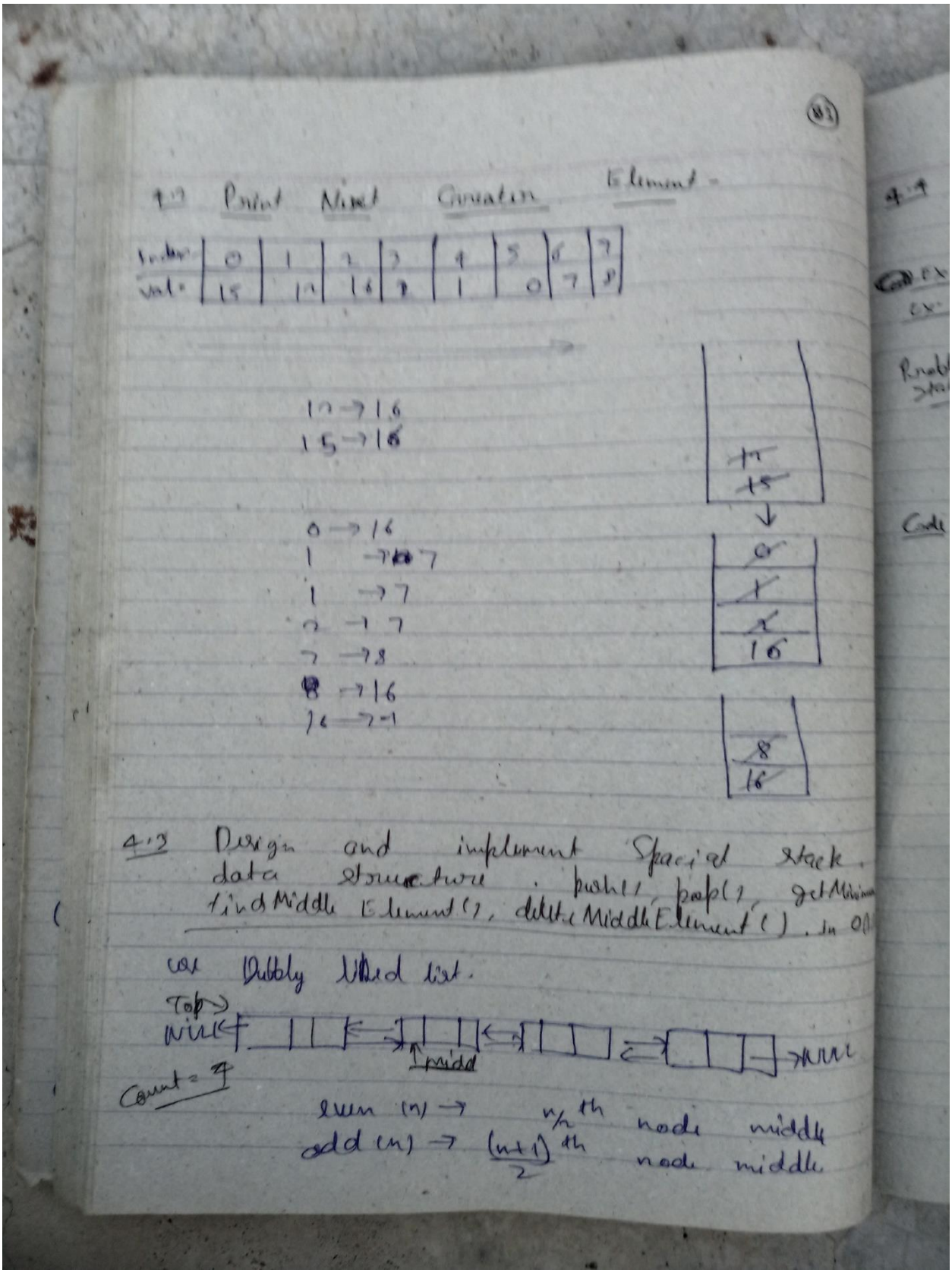
push(x) – Push element x onto stack
pop() – Remove the element on top of the stack.
top() – Get the top element
getMin() – Retrieve the minimum element in the stack

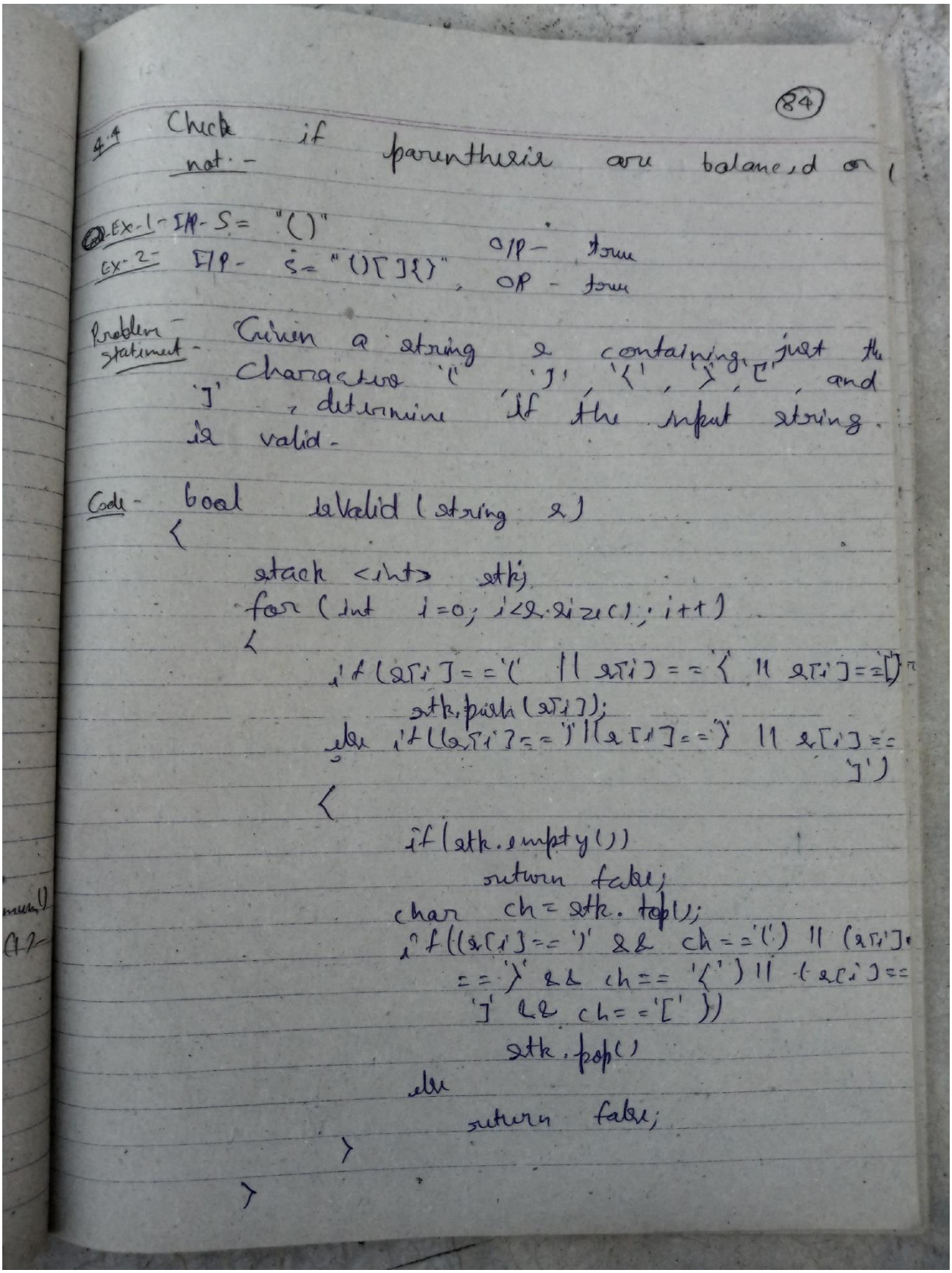
Code –

```
#include<bits/stdc++.h>
using namespace std;
int minEle;
stack<int> s;
void push(int ele)
{
    if(s.empty())
    {
        s.push(ele);
        minEle = ele;
    }
    else
    {
        if(ele >= minEle)
            s.push(ele);
        else if(ele < minEle)
        {
            s.push(2*ele - minEle);
            minEle = ele;
        }
    }
}
```



```
void pop()
{
    if(s.empty())
        cout << "Empty" << endl;
    else
    {
        if(s.top() >= minEle)
            s.pop();
        else if(s.top() < minEle)
        {
            minEle = 2*minEle - s.top();
            s.pop();
        }
    }
}
int top()
{
    if(s.empty())
        return -1;
    else
    {
        if(s.top() >= minEle)
            return s.top();
        else if(s.top() < minEle)
            return minEle;
    }
}
int getMin()
{
    if(s.empty())
        return -1
    return minEle;
}
```


Next Greater Element
Index
val
17 → 16
15 → 16
0 → 16
1 → 7
1 → 7
2 → 7
7 → 8
8 → 16
16 → -1
4.3 Design and implement Special stack. data structure. push(), pop(), getMinimum(), findMiddle Element(), deleteMiddleElement(). In O(1)
use Doubly linked list.
Top
NULL
middle
Count = 7
even (n) → n/2 th node middle
odd (n) → (n+1)/2 th node middle



(84)

4.4 Check if parenthesis are balanced or not -

Ex-1 - I/P - S = "()" O/P - true

Ex-2 - I/P - S = "()[]{}" O/P - true

Problem statement - Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

Code -

```
bool isValid(string s)
{
    stack<int> stk;
    for(int i=0; i<s.size(); i++)
    {
        if(s[i]=='(' || s[i]=='{' || s[i]=='[')
            stk.push(s[i]);
        else if((s[i]==')' || s[i]=='}' || s[i]==']')
        {
            if(stk.empty())
                return false;
            char ch = stk.top();
            if((s[i]==')' && ch=='(') || (s[i]=='}' && ch=='{') || (s[i]==']' && ch=='['))
                stk.pop();
            else
                return false;
        }
    }
```

(85)

```
if (stk.empty())
    return true;
return false;
}
```

4.5 Stack Span Problem

Prices = { 100, 80, 60, 70, 60, 75, 85 }
0 1 2 3 4 5 6

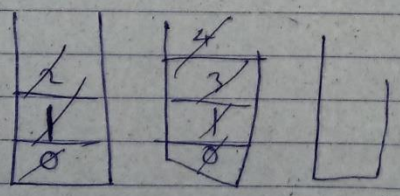
O/P - { 1, 1, 1, 2, 1, 4, 6 }

Code-

Use Stack

P = { 100, 80, 60, 70, 60, 75, 85 }
0 1 2 3 4 5 6

Span = { 1, 1, 1, 3-1, 4-3, 5-1, 6-



while P[stack[top]] >

pop

i - stack[top]

4.6 The Celebrity Problem

Party : $V_0, V_1, V_2, \ldots, V_{n-1}$

Rules:-
1- V_3 is a celebrity if V_0, V_1 & V_2 all of them know V_3
2- V_3 does not know V_0 V_1 V_2

86

I/p -

$$A = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

(Adjacency Matrix)

knows $A_{ij} = \begin{cases} 1, & \text{if } V_i \text{ knows } V_j \\ 0, & \end{cases}$

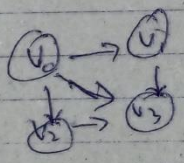
O/P - 3 | Task - Find the celebrity, if he/she exists.

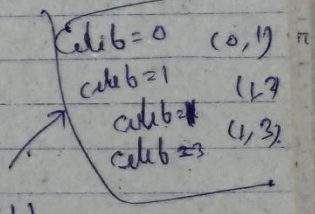
Code -

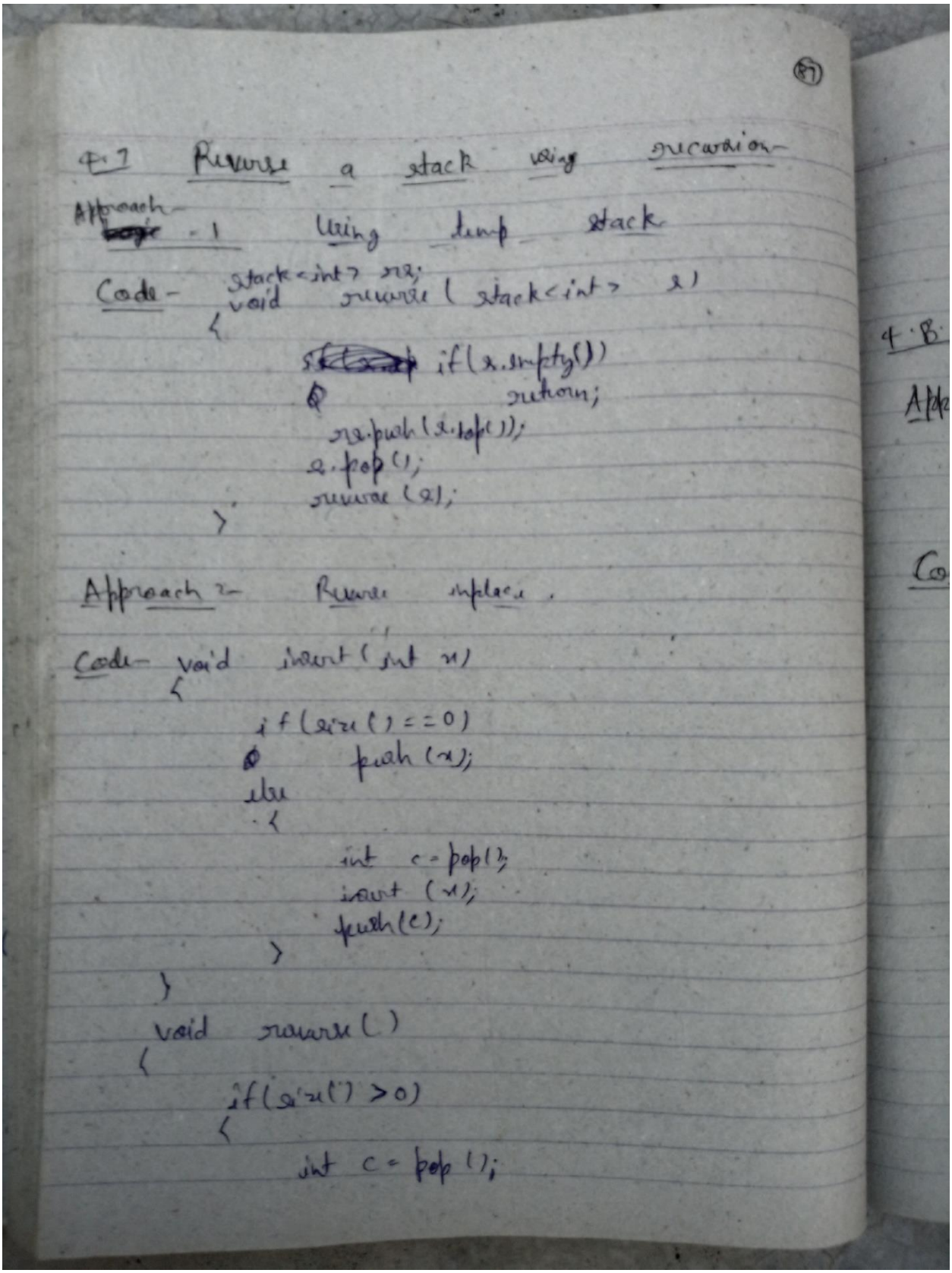
```
int matrix[3][3] = {{0, 1, 0},
                    {0, 0, 0},
                    {0, 1, 0}};

int Knows(int a, int b)
{
    return matrix[a][b];
}

int main(void)
{
    int celeb = 0, n = 3;
    for(int i=0; i<n; i++)
        if(Knows(celeb, i) == 1)
            celeb = i;
    for(int i=0; i<n; i++)
        if((i != celeb) && (Knows(celeb, i) == 1 ||
                            Knows(i, celeb) != 1))
            printf("-1");
    printf("%d", celeb);
    return 0;
}
```







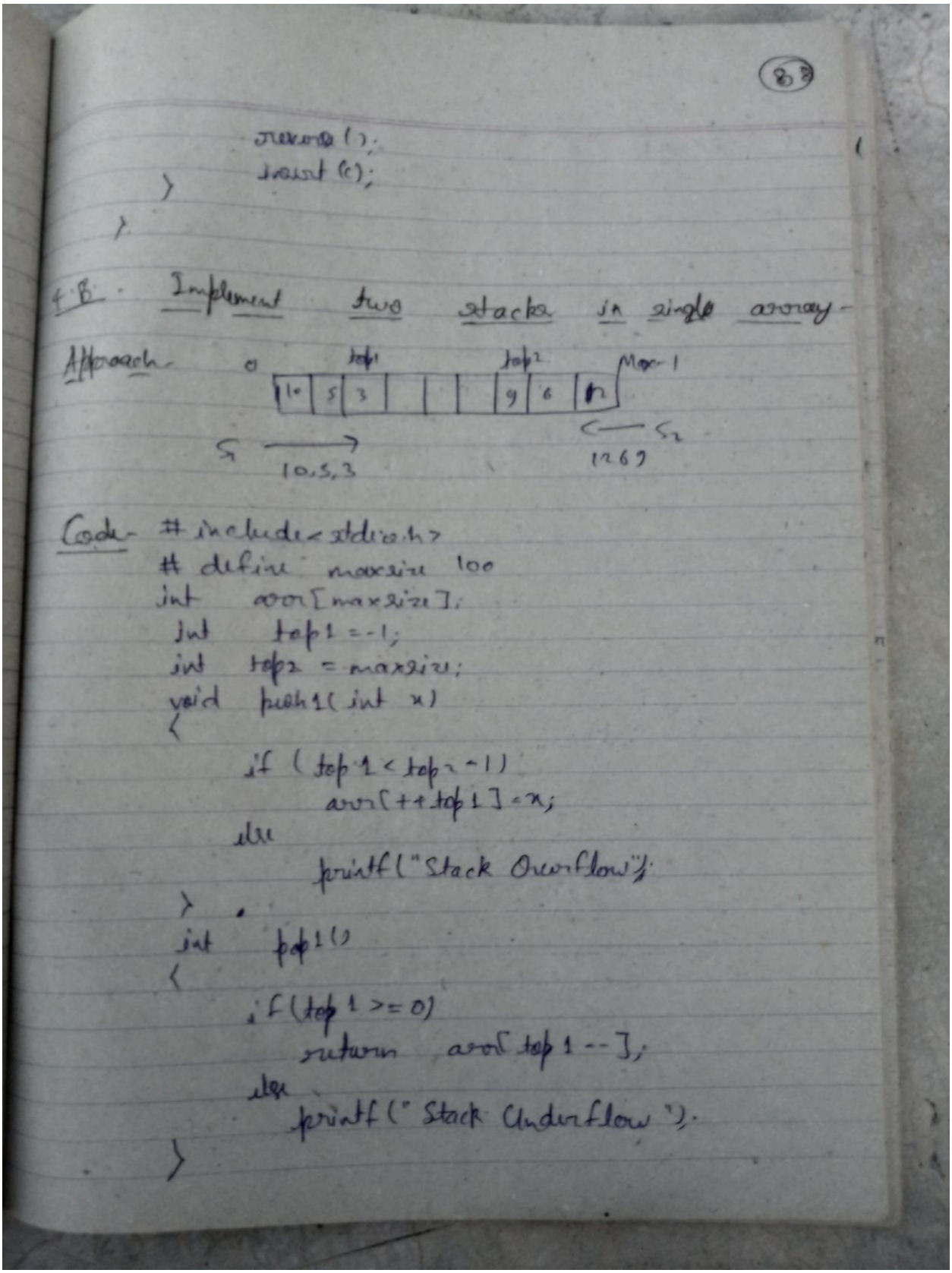
Q.7 Reverse a stack using recursion

Approach - 1 Using temp stack

Code -
```
stack<int> s2;
void reverse(stack<int> s)
{
    if(s.empty())
        return;
    s2.push(s.top());
    s.pop();
    reverse(s);
}
```

Approach 2 - Reverse inplace.

Code -
```
void insert(int x)
{
    if(size()==0)
        push(x);
    else
    {
        int c = pop();
        insert(x);
        push(c);
    }
}
void reverse()
{
    if(size()>0)
    {
        int c = pop();
```



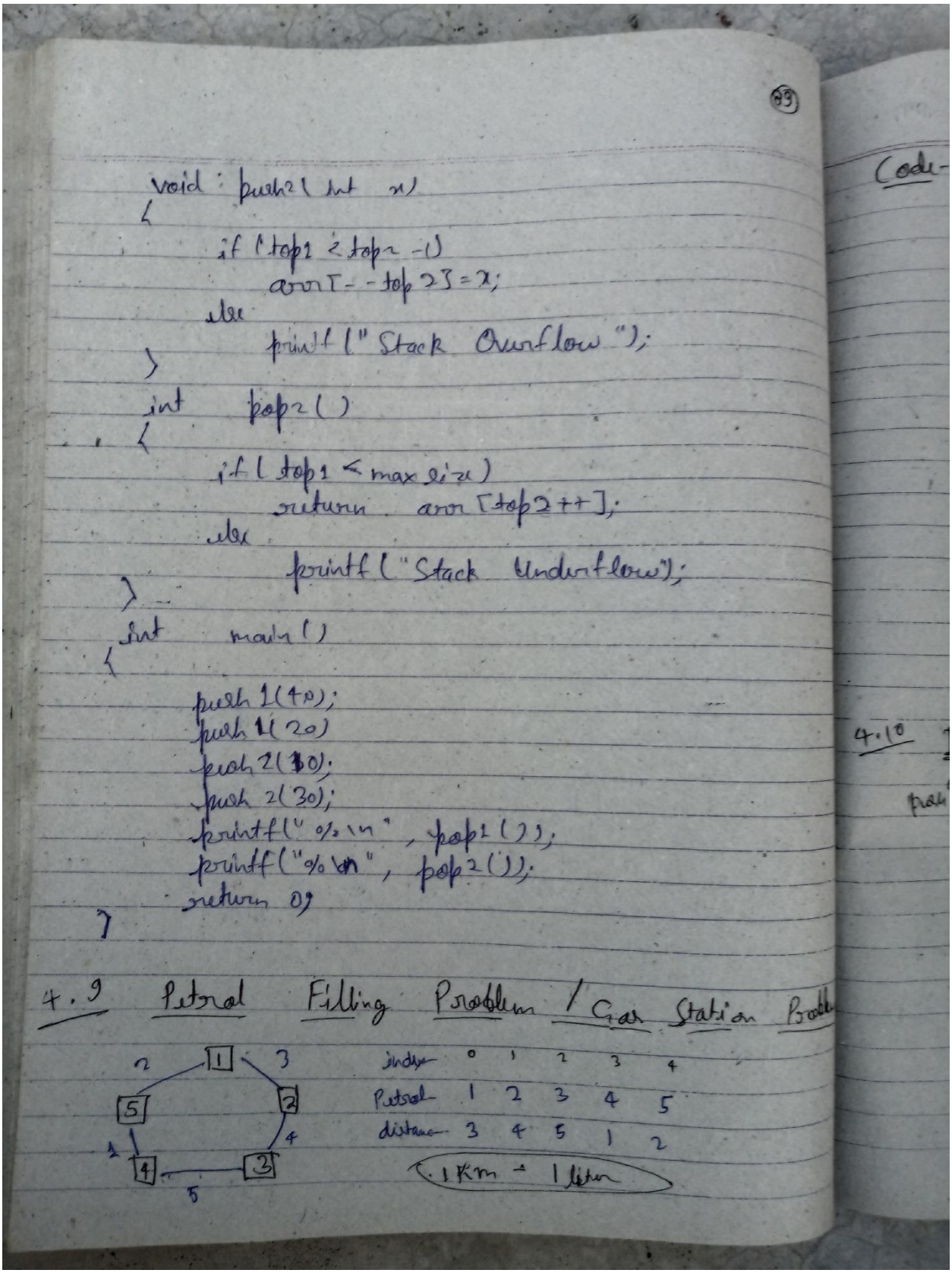
```
        pop();
        print(c);
    }
}
```

4. B. Implement two stacks in single array-

Approach-

Code-

```
#include<stdio.h>
#define maxsize 100
int arr[maxsize];
int top1 = -1;
int top2 = maxsize;
void push1(int x)
{
    if (top1 < top2 - 1)
        arr[++top1] = x;
    else
        printf("Stack Overflow");
}
int pop1()
{
    if(top1 >= 0)
        return arr[top1--];
    else
        printf("Stack Underflow");
}
```



```
void push2(int x)
{
    if(top2 > top1-1)
        arr[--top2] = x;
    else
        printf("Stack Overflow");
}
int pop2()
{
    if(top2 < maxsize)
        return arr[top2++];
    else
        printf("Stack Underflow");
}
int main()
{
    push1(10);
    push1(20);
    push2(10);
    push2(30);
    printf("%d\n", pop1());
    printf("%d\n", pop2());
    return 0;
}
```

4.9 Petrol Filling Problem / Gas Station Prob

index	0	1	2	3	4
Petrol	1	2	3	4	5
distance	3	4	5	1	2

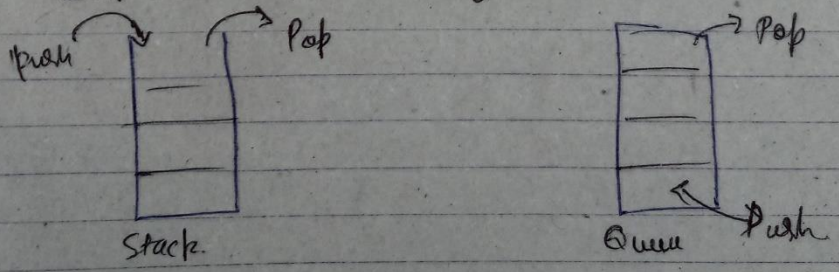
1 Km = 1 liter

$T(n) = O(n)$ 90

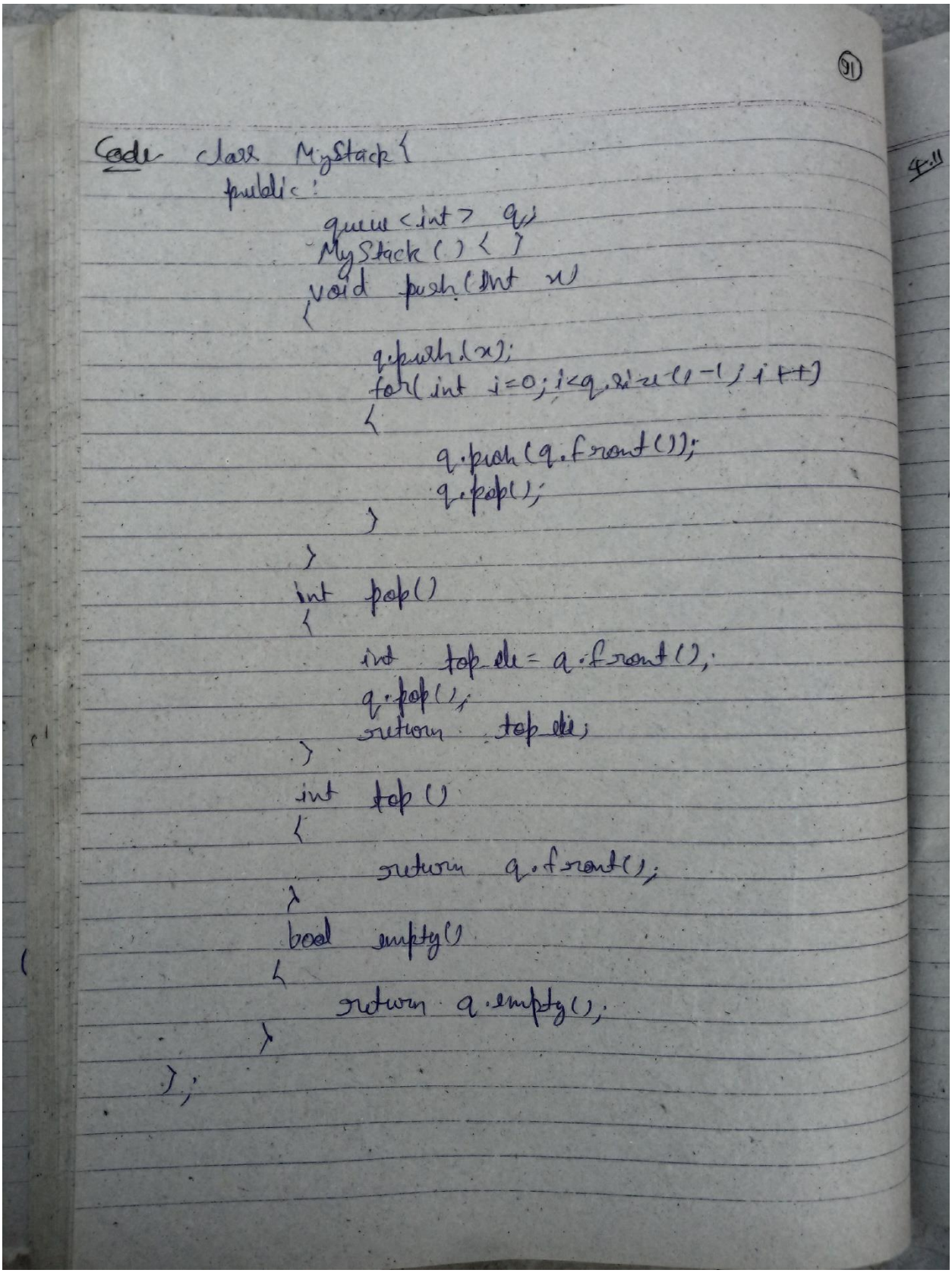
Code -

```
int canCompleteCircuit(vector<int>& gas,
                       vector<int>& cost)
{
    int sum = 0, start = 0, diff = 0;
    for (int i = 0; i < gas.size(); i++)
    {
        sum = sum + gas[i] - cost[i];
        if (sum < 0)
        {
            start = i+1;
            diff += sum;
            sum = 0;
        }
    }
    return sum + diff >= 0 ? start : -1;
}
```

4.10 Implement stack using queue -



Pop - O(n)
Push - O(n).

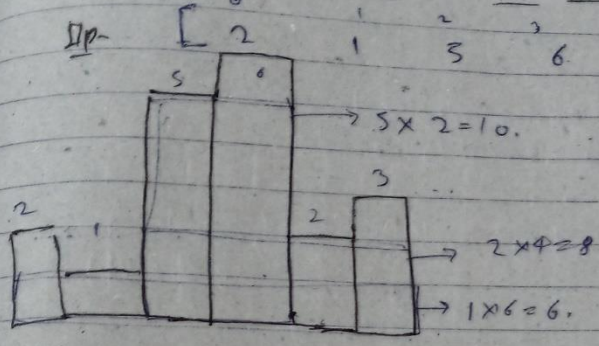


```
Code  class MyStack {
      public:
          queue<int> q;
          MyStack() { }
          void push(int x)
          {
              q.push(x);
              for(int i=0; i<q.size()-1; i++)
              {
                  q.push(q.front());
                  q.pop();
              }
          }
          int pop()
          {
              int top_ele = q.front();
              q.pop();
              return top_ele;
          }
          int top()
          {
              return q.front();
          }
          bool empty()
          {
              return q.empty();
          }
      };
```

(92)

Largest Rectangle in Histogram.

Ip- [2 1 5 6 2 3]



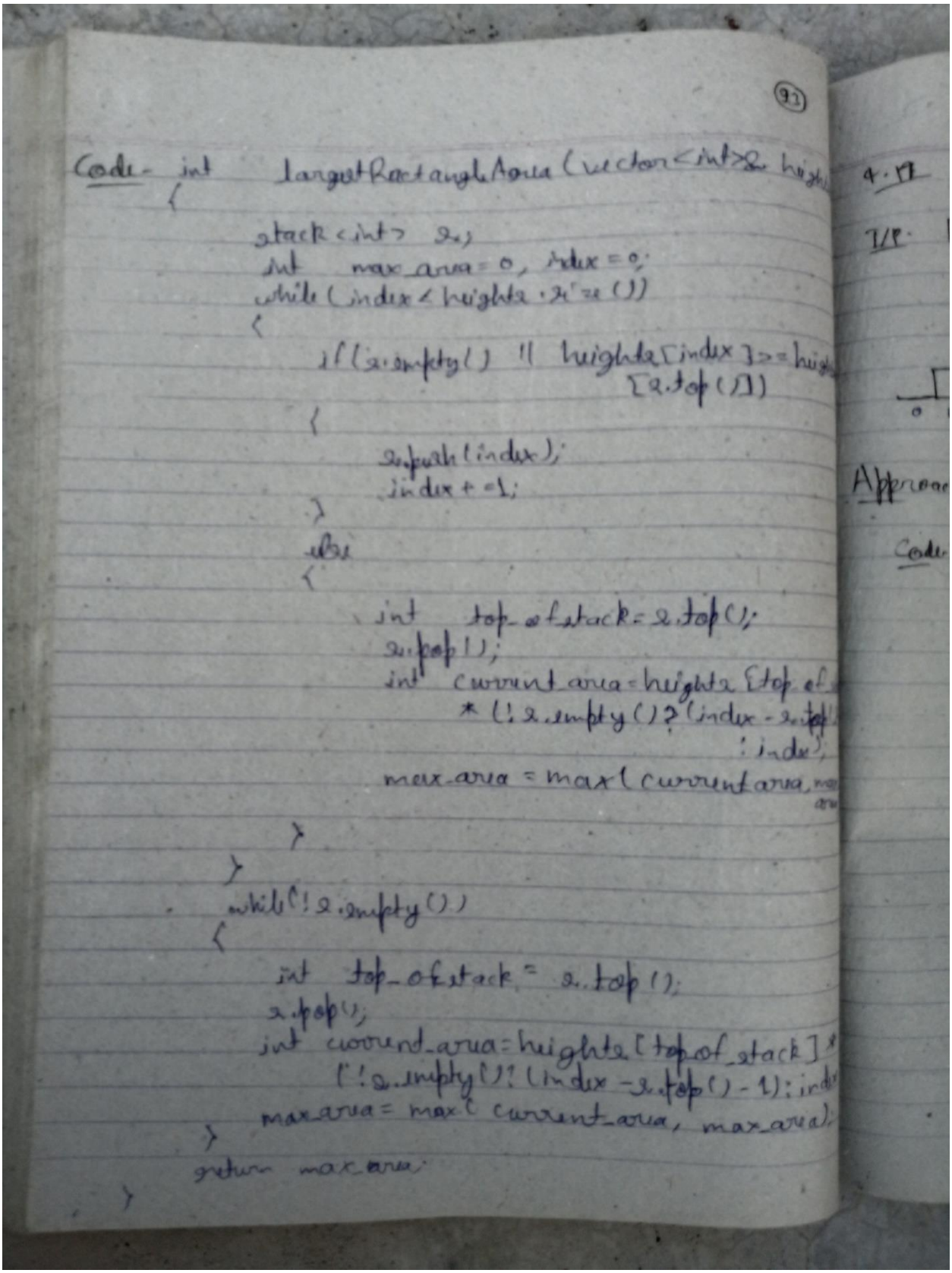
T(n) = O(n)

S(n) = O(n)

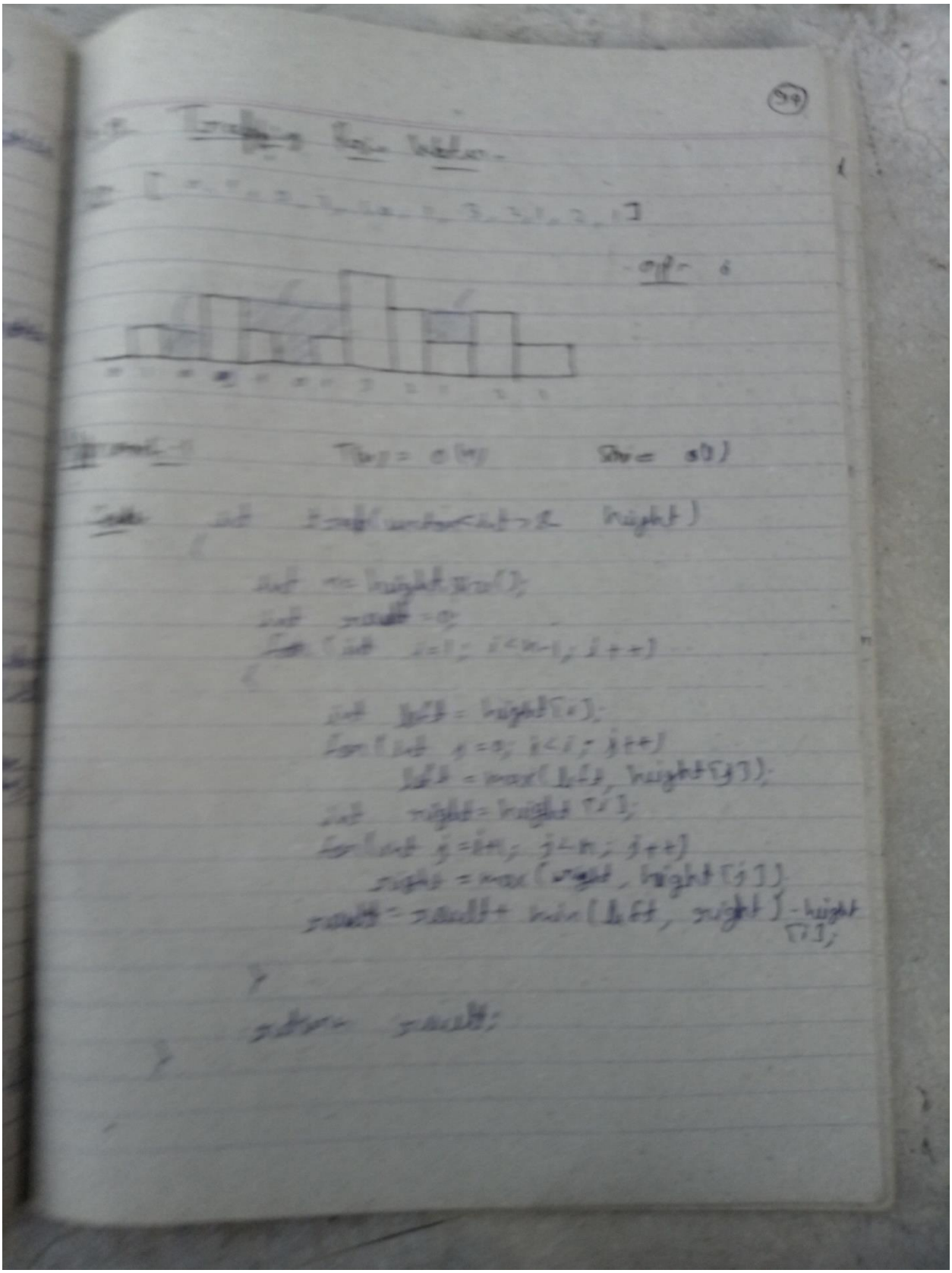
Op = 10

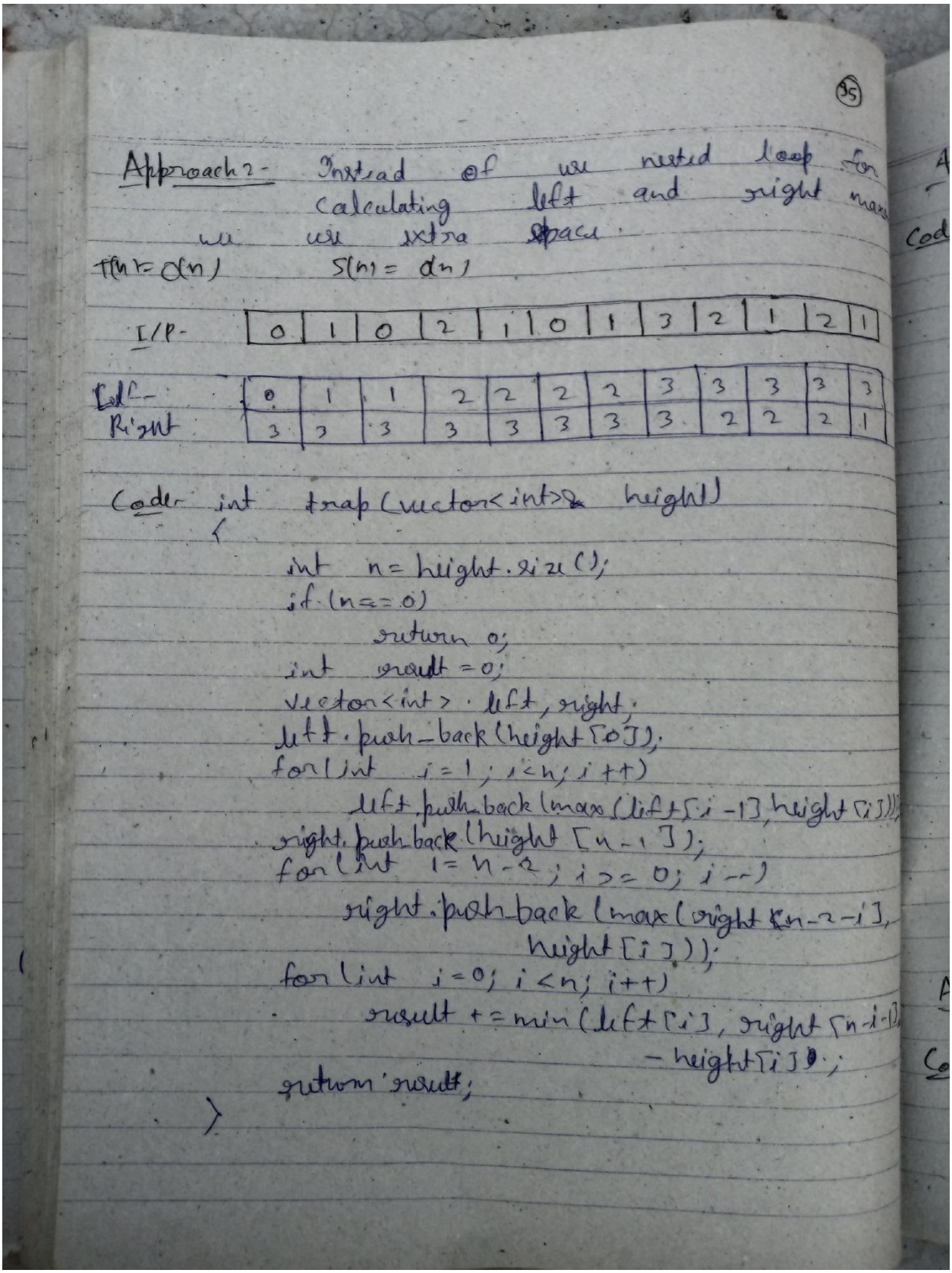
Algorithm

1) Create an empty stack
2) Traverse every element present in an array / list.
   (a) If stack is empty (0 0) current[i] height is greater than top of the stack element then push the current index into the stack.
   (b) Use pop removing the top of stack while top of the stack is greater.
3) If stack is empty, then one by one remove all bars from stack repeat step 2.



```
Code- int largestRectangleArea(vector<int>& heights)
{
    stack<int> s;
    int max_area = 0, index = 0;
    while (index < heights.size())
    {
        if(s.empty() || heights[index] >= heights[s.top()])
        {
            s.push(index);
            index += 1;
        }
        else
        {
            int top_of_stack = s.top();
            s.pop();
            int current_area = heights[top_of_stack]
                * (!s.empty() ? (index - s.top()
                                 : index);
            max_area = max(current_area,
        }
    }
    while(!s.empty())
    {
        int top_of_stack = s.top();
        s.pop();
        int current_area = heights[top_of_stack] *
            (!s.empty() ? (index - s.top() - 1) : ind
        max_area = max(current_area, max_area);
    }
    return max_area;
}
```





(35)

Approach 2- Instead of calculating left and right max we use nested loop for we use extra space.

T(n) = O(n) S(n) = O(n)

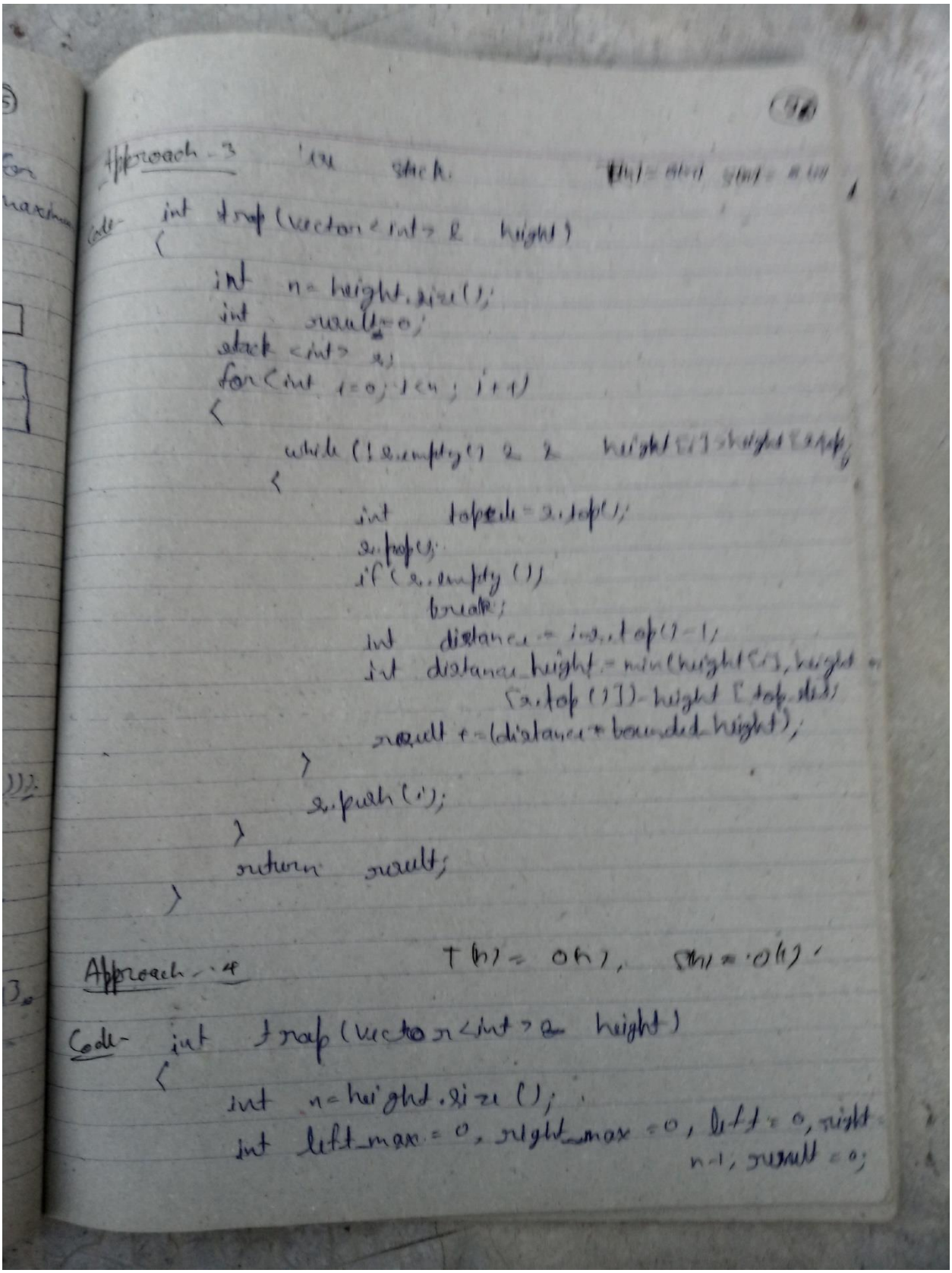
I/P-

0	1	0	2	1	0	1	3	2	1	2	1

Left-	0	1	1	2	2	2	2	3	3	3	3	3
Right	3	3	3	3	3	3	3	3	2	2	2	1

Code-

```
int trap(vector<int>& height)
{
    int n = height.size();
    if (n == 0)
        return 0;
    int result = 0;
    vector<int> left, right;
    left.push_back(height[0]);
    for(int i = 1; i < n; i++)
        left.push_back(max(left[i-1], height[i]));
    right.push_back(height[n-1]);
    for(int i = n-2; i >= 0; i--)
        right.push_back(max(right[n-2-i],
                            height[i]));
    for(int i = 0; i < n; i++)
        result += min(left[i], right[n-i-1])
                  - height[i];
    return result;
}
```

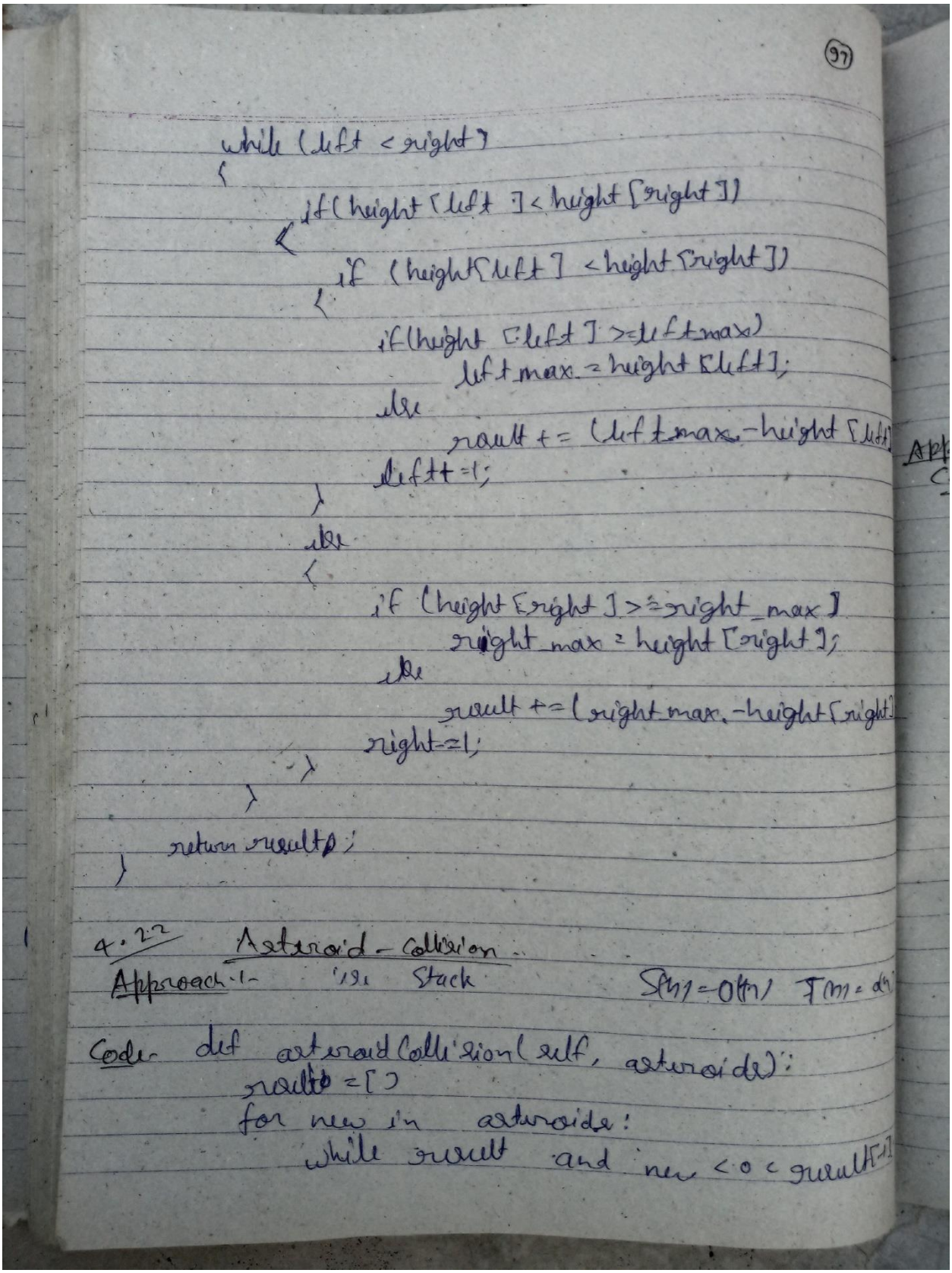


Approach - 3 using stack.

```
int trap(vector<int>& height)
{
    int n = height.size();
    int result = 0;
    stack<int> s;
    for(int i=0; i<n; i++)
    {
        while(!s.empty() && height[i] > height[s.top()])
        {
            int top = s.top();
            s.pop();
            if(s.empty())
                break;
            int distance = i - s.top() - 1;
            int distance_height = min(height[i], height
                [s.top()]) - height[top];
            result += (distance * bounded_height);
        }
        s.push(i);
    }
    return result;
}
```

Approach - 4

```
int trap(vector<int>& height)
{
    int n = height.size();
    int left_max = 0, right_max = 0, left = 0, right =
                                    n-1, result = 0;
```



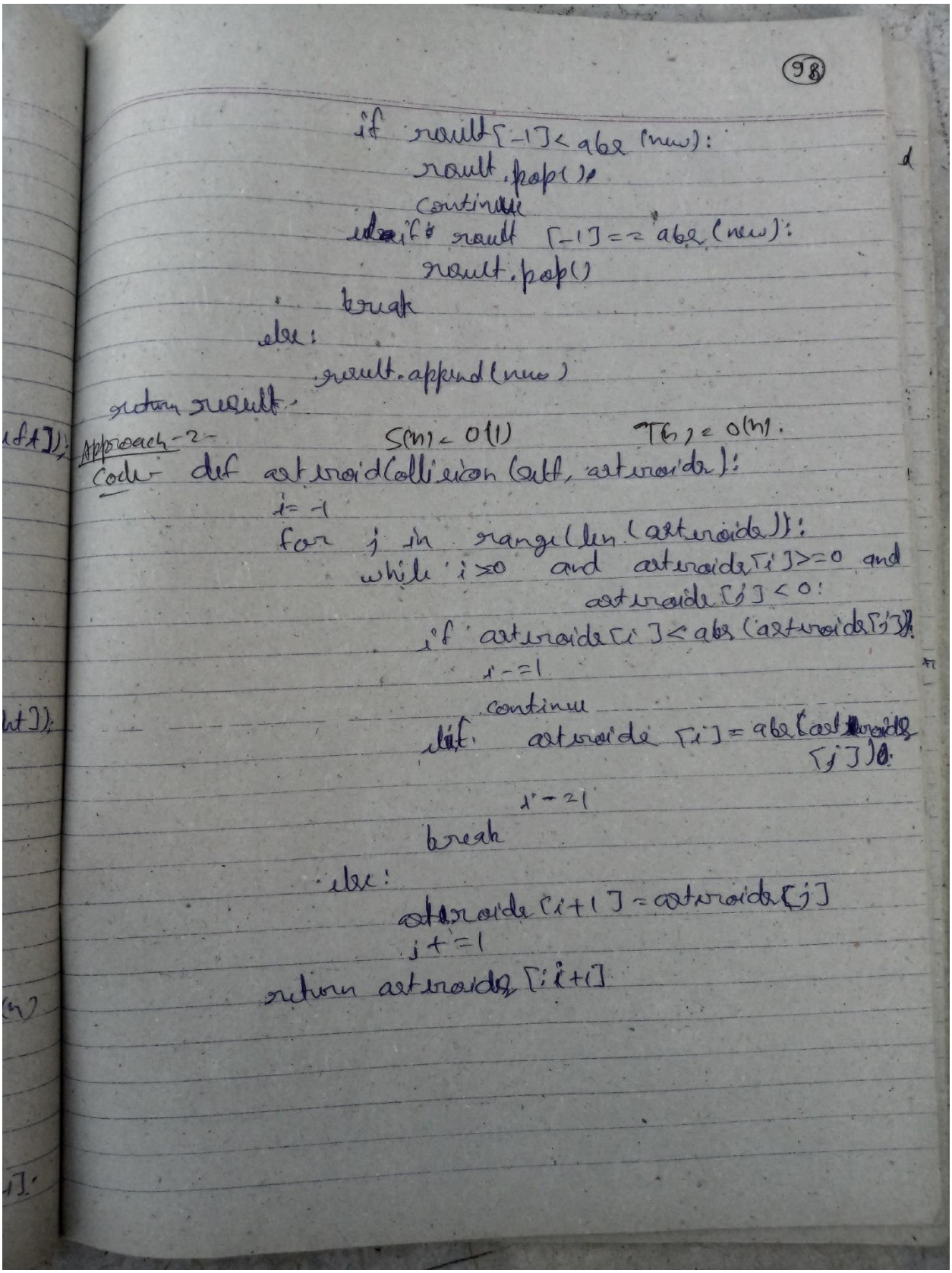
```
while (left < right)
{
    if (height[left] < height[right])
    {
        if (height[left] < height[right])
        {
            if (height[left] >= left_max)
                left_max = height[left];
            else
                result += (left_max - height[left]);
            left++;
        }
        else
        {
            if (height[right] >= right_max)
                right_max = height[right];
            else
                result += (right_max - height[right]);
            right--;
        }
    }
    return result;
}
```

4.22 Asteroid Collision

Approach 1- Using Stack

Code-

```
def asteroidCollision(self, asteroids):
    result = []
    for new in asteroids:
        while result and new < 0 < result[-1]:
```



98

```
if result[-1] < abs(num):
    result.pop()
    continue
elif result[-1] == abs(num):
    result.pop()
    break
else:
    result.append(num)
return result
```

Approach-2 S(n) = O(1) T(n) = O(n).

Code

```
def asteroidCollision(self, asteroids):
    i = -1
    for j in range(len(asteroids)):
        while i >= 0 and asteroids[i] >= 0 and
                asteroids[j] < 0:
            if asteroids[i] < abs(asteroids[j]):
                i -= 1
                continue
            elif asteroids[i] == abs(asteroids[j]):
                i -= 1
                break
            else:
                asteroids[i+1] = asteroids[j]
                i += 1
    return asteroids[:i+1]
```

99

Problems on Tree

5.1 Count number of nodes in a tree (Recursive Code)

Approach - Like recursion

TM = O(n) ; worst case SM = O(n). (stack)

```
Code- int countNodes(TreeNode * root)
{
    return root ? countNodes(root->left) +
                  countNodes(root->right) : 0;
}
```

Approach 2 - Use iterative approach. TM = O(n)

5.2 Check if two trees are identical or not

Approach 1 - Recursive Approach; TM = O(h)

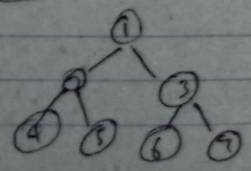
```
Code- bool isSameTree(TreeNode * p, TreeNode * q)
{
    if(!p && !q)
        return true;
    if(p && q)
        return (p->val == q->val) && isSameTree
            (p->left, q->left) && isSameTree(
            p->right, q->right);
    return false;
}
```

Approach 2 - Iterative. TM = O(n).

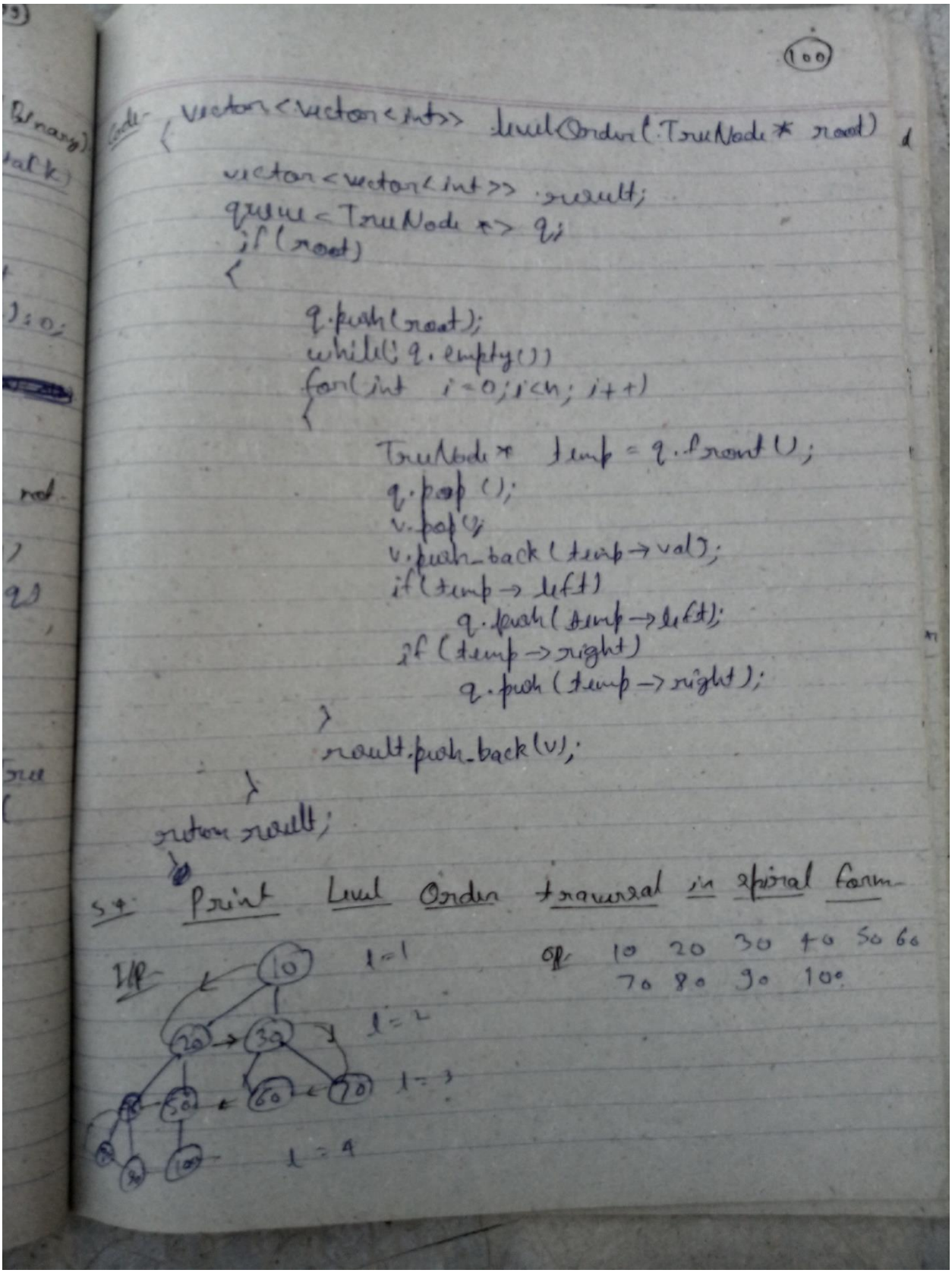
5.3 Level Order Tree Traversal -

Approach - BFS (Queue). TM = O(n).

I/P



O/P 1, 2, 3, 4, 5, 6, 7



100

Code:

```
vector<vector<int>> levelOrder(TreeNode* root) {
    vector<vector<int>> result;
    queue<TreeNode*> q;
    if(root)
    {
        q.push(root);
        while(!q.empty())
        for(int i=0;i<n;i++)
        {
            TreeNode* temp = q.front();
            q.pop();
            v.pop();
            v.push_back(temp->val);
            if(temp->left)
                q.push(temp->left);
            if(temp->right)
                q.push(temp->right);
        }
        result.push_back(v);
    }
return result;
```

Q: Print Level Order traversal in spiral form

I/P

O/P 10 20 30 40 50 60 70 80 90 100