09/10/2021

Problem Solving

Applied Prep Course

SATYAM SETH

PART-2

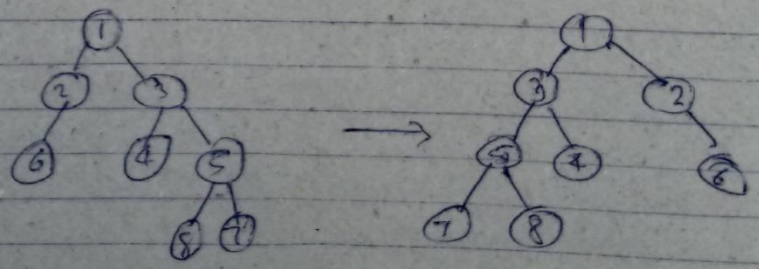
$T(n) = O(n)$ $S(n) = O(n)$.

Approach- Use 2 stacks one for odd level and another for even level.

Algo-

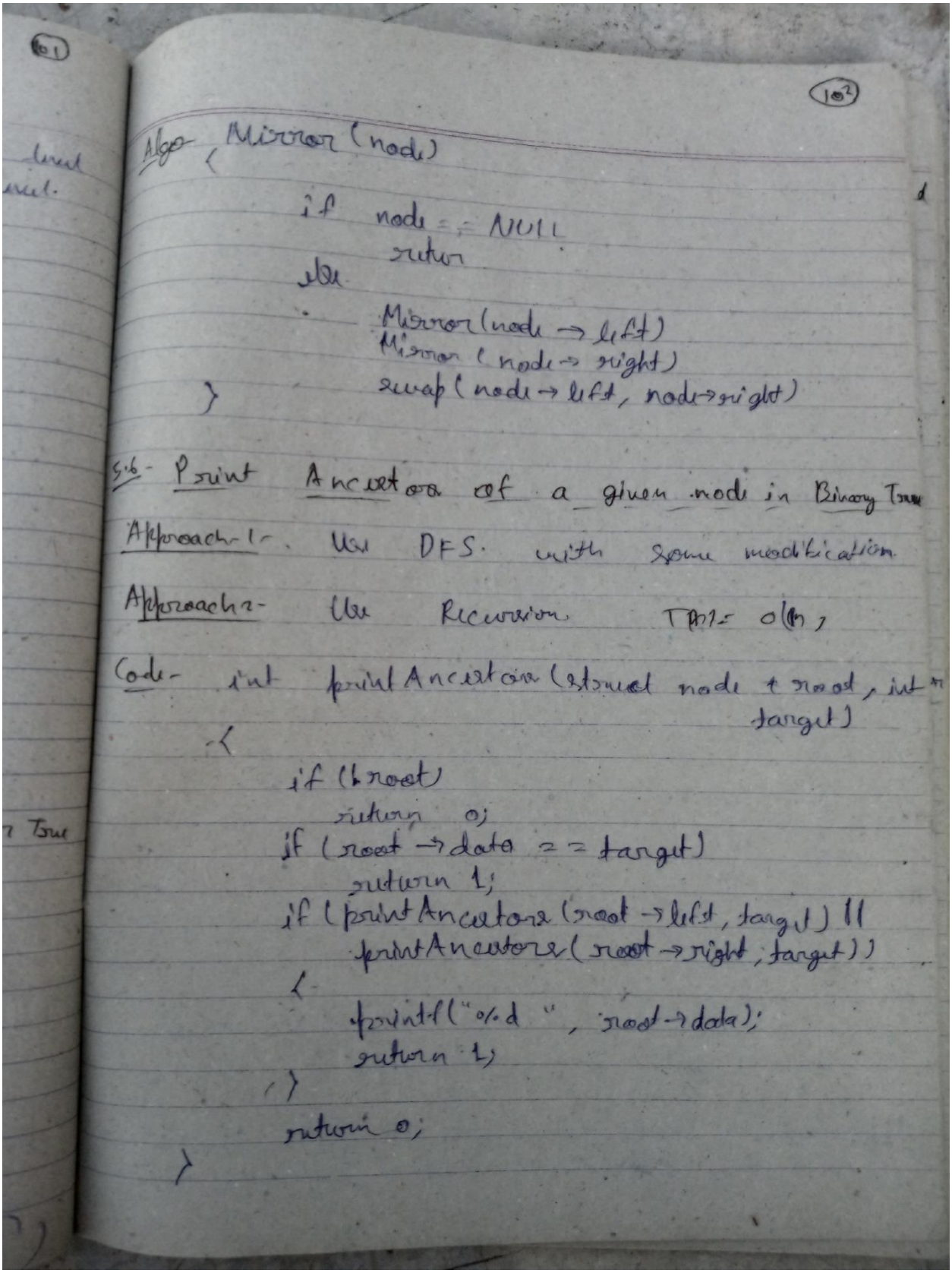
```
s1.push(root)
while !isEmpty(s1) & !isEmpty(s2)
    while !isEmpty(s1)
        n = s1.pop()
        print n->val.
        if n->right is not null.
            s2.push(n->right)
        if n->left is not null
            s2.push(n->left)
    while !isEmpty(s2)
        n = s2.pop()
        print n->val
        if n->left is not NULL
            s1.push(n->left)
        if n->right is not null
            s1.push(n->right)
```

5.5- Convert a Binary Tree into its Mirror Tree OR Invert Binary Tree-



Approach- Use recursion

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



Algo- Mirror (node)
{
if node == NULL
return
else
Mirror(node → left)
Mirror (node → right)
swap (node → left, node → right)
}

5.6- Print Ancestors of a given node in Binary Tree

Approach-1- Use DFS with some modification

Approach-2- Use Recursion TC:- O(n)

Code- int printAncestors (struct node * root, int target)
{
if (!root)
return 0;
if (root → data == target)
return 1;
if (printAncestors (root → left, target) ||
printAncestors (root → right, target))
{
printf("%d ", root → data);
return 1;
}
return 0;
}

(103)

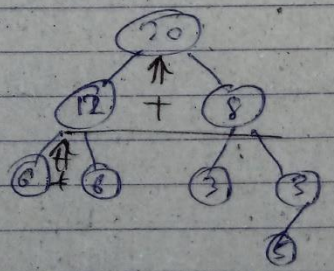
Note. Compiler convert tail recursion into iterative

5.7- Find Lowest Common Ancestor in a Binary Search Tree-

Approach- Use Recursion. T(n) = O(h) S(n) = O(1).

Algo- lca(node, x, y)
{
if node == NULL
return NULL
if (node->x > x && node > y)
return lca(node->left, x, y)
else if (node x < x && node < y)
return lca(node->right, x, y)
else
return node.
}

5.8- Children sum property in a binary tree-



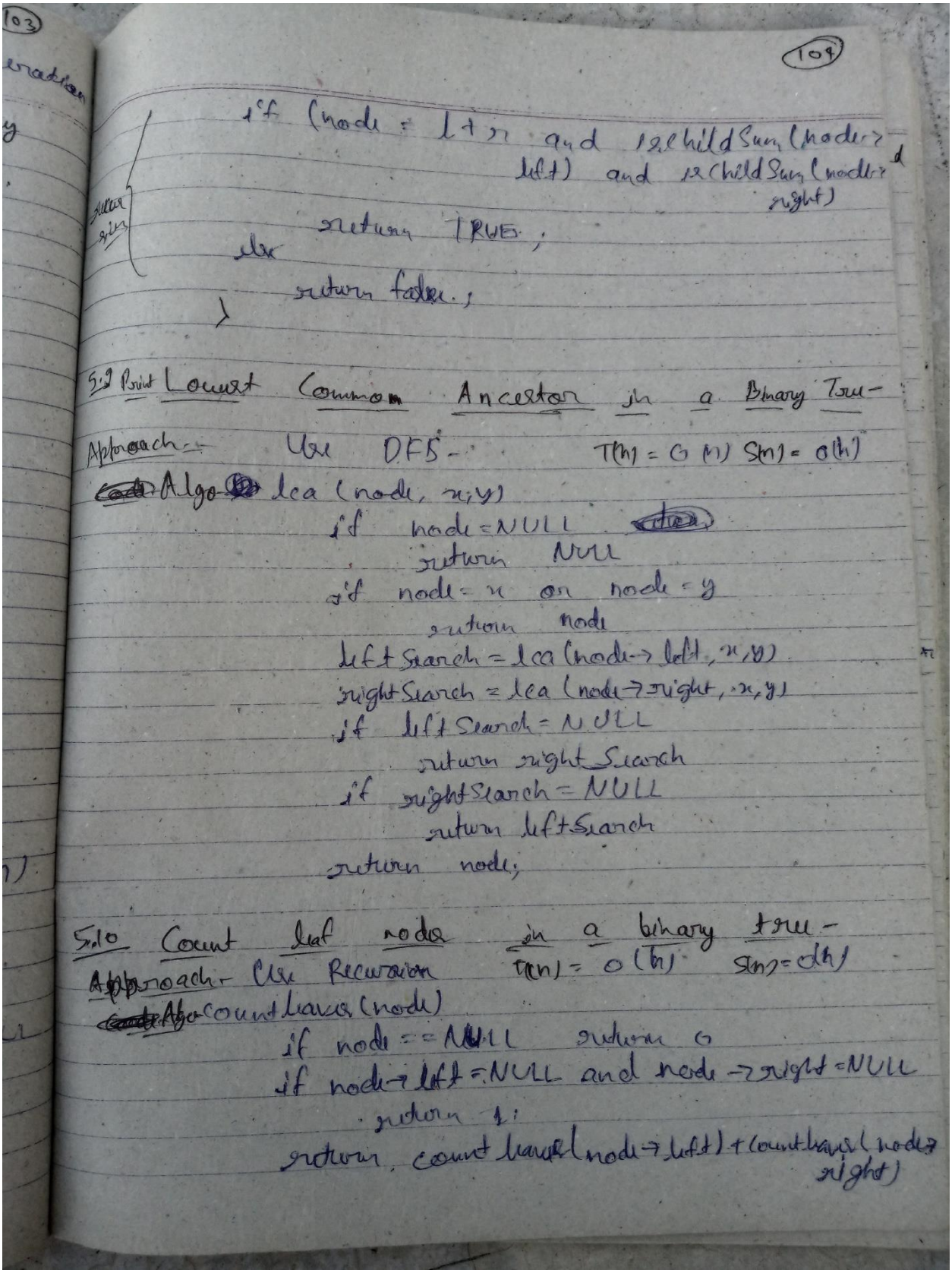
12+8=20. True

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

Approach- Use Recursion

Algo- isChildSum(node)
{
base { if (node == NULL) or (node->right = NULL & node->left = NULL)
return TRUE

check { l = 0; r = 0;
if node->left is not NULL
l = node->left.
if node->right is not NULL
r = node->right



103

104

if (node = left or right and isChildSum(node->left) and isChildSum(node->right)

return TRUE;

else

return false;

}

5.9 Print Lowest Common Ancestor in a Binary Tree-

Approach:- Use DFS- T(n) = O(n) S(n) = O(h)

Algo lca(node, x, y)

if node = NULL return

return NULL

if node = x or node = y

return node

leftSearch = lca(node->left, x, y)

rightSearch = lca(node->right, x, y)

if leftSearch = NULL

return rightSearch

if rightSearch = NULL

return leftSearch

return node;

5.10 Count leaf nodes in a binary tree-

Approach- Use Recursion T(n) = O(h) S(n) = O(h)

Algo countLeaves(node)

if node == NULL return 0

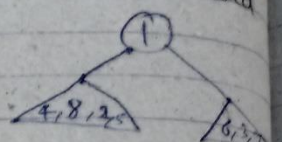
if node->left = NULL and node->right = NULL

return 1;

return countLeaves(node->left) + countLeaves(node->right)

105

5.11 Construct a binary tree from inorder and post order traversals.

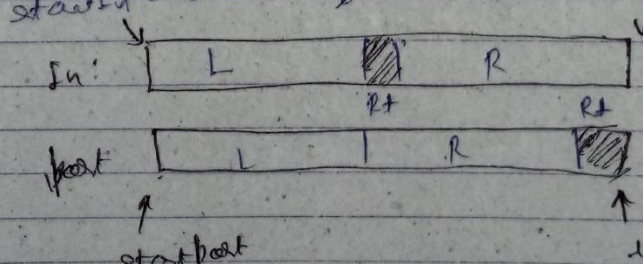


In: 4, 8, 2, 5, 1, 6, 3, 7

Post: 8, 4, 5, 2, 6, 7, 3, 1

Algo- buildTree(in, Post, startIn, endIn, StartPost, endPost)

```
if startIn < endIn
    return NULL
if startIn = endIn
    return makeNode(in[startIn])
n = makeNode(Post[endPost])
idx = search(in, startIn, endIn, Post[endPost])
n->left = buildTree(in, post, startIn, idx-1, startIn, idx-1)
n->right = buildTree(in, post, idx+1, endIn, idx, endIn-1)
return n
```



If we use linear search then

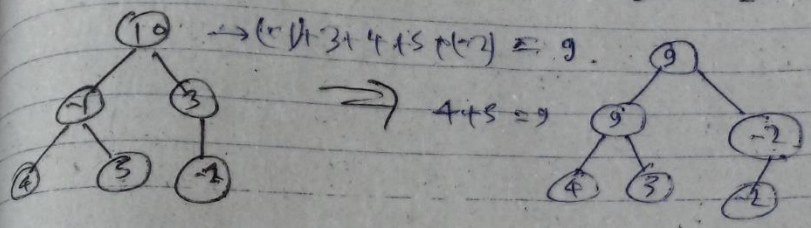
$T(n) = O(n^2)$, $S(n) = O(n)$.

If we use hash map : (key = nodevalue, value = index) then

$T(n) = O(n)$, $S(n) = O(n)$

(106)

Q5 Convert a given tree to its Sum Tree -

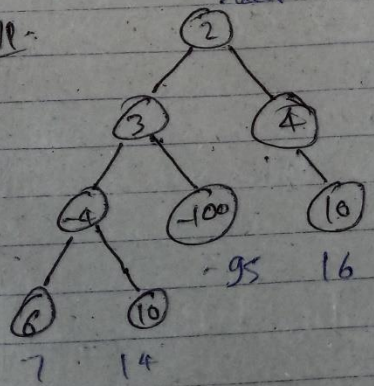


Approach- Use DFS. TM1= O(n) SM1= O(h)

Algo- int SumTree(node)
if node == NULL
return 0.
l = SumTree(node->left)
r = SumTree(node->right
v = node->val;
node->val = l+r;
return l+r+v

Q.B Find the maximum sum leaf to root path in a Binary Tree -

I/P- root



O/P- 2 → 4 → 10

Approach- Use DFS. TM= O(n) , SM= O(h)

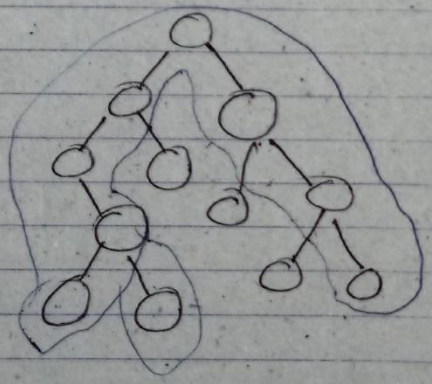
(107)

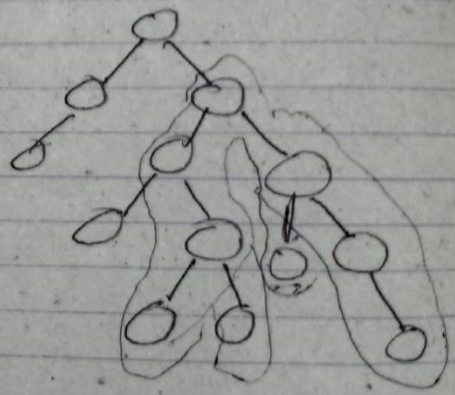
Algo- maxSumLeaf (node, currSum, bestSum, bestNode)

base case:
- if node == NULL return NULL
- currSum = currSum + node → val
- if node → left = NULL & node → right = NULL
  - if currSum > bestSum
    - bestNode = node;

recursive DFS:
- maxSumLeaf (node → left, currSum, bestSum, bestNode)
- maxSumLeaf (node → right, currSum, bestSum, bestNode)

5.14 Find Diameter of a Binary Tree-



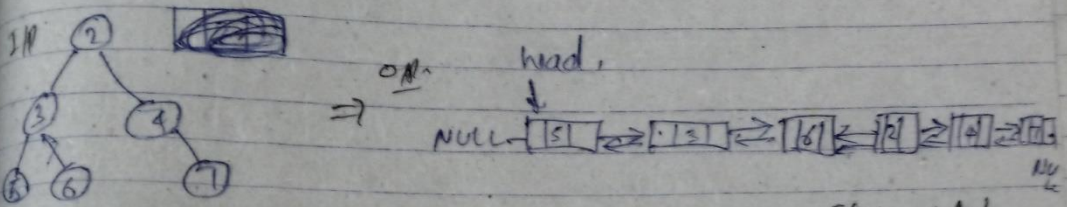


Approach- We find height using recursion (height (LS) + height (RS) + 2). T(n) = O(n) S(n) = O(h)

Algo-
```
diam = -∞
height (node)
    if node = Null return (-1);
    l = height (node → left)
    r = height (node → right)
    diam = max(diam, l+r+2)
    return (1 + max (l, r))
```

(108)

Q.15 Convert a given Binary Tree to Doubly Linked List.



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

Approach - Use Morder traversal.

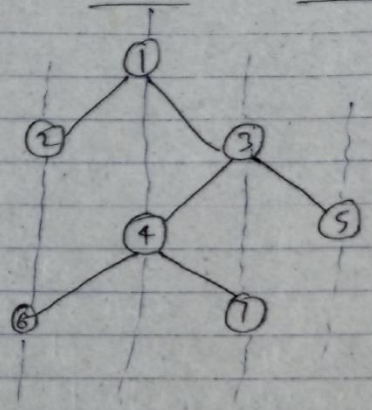
Code - void BinaryToDLL (struct node* root, struct node** head)

```
{
// Base case
if (!root)
    return;
// Recursively convert right subtree
BinaryToDLL (root->right, head);
// insert root into DLL
root->right = *head;
// change left pointer of previous head
if (*head)
    (*head)->left = root;
// change head of Doubly linked list
*head = root;
// Recursively convert left subtree
BinaryToDLL (root->left, head);
}
```

(109)

5.16 Vertical traversal of binary tree -

I/P.

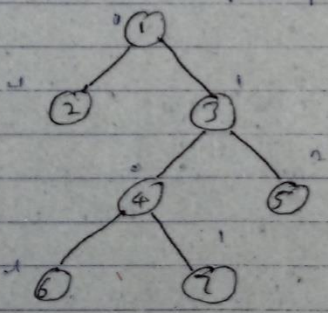


O/P

Vertical order.

2, 6
1, 4
3, 7
5

Approach: Use traversal algo with hashmap.



Hash Map.

Key		Values (Array)
-1	→	2, 6
0	→	1, 4
1	→	3, 7
2	→	5

Algo

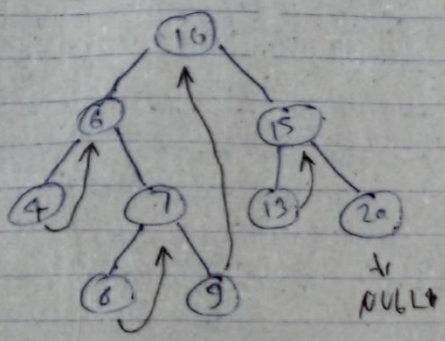
```
verticalorder(Node *p, dist, Hashmap* h)
{
    if p == NULL
        return
    h.insert(dist, p)
    verticalOrder(p->left, dist-1, h)
    verticalOrder(p->right, dist+1, h)
}
```

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

(110)

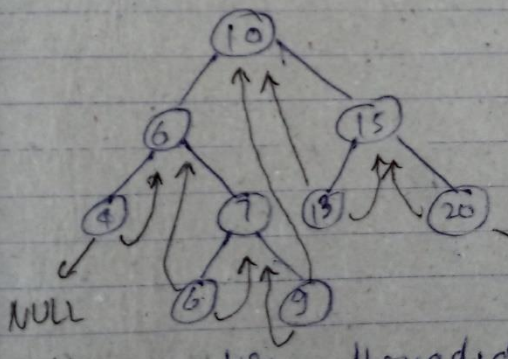
5.11 Inorder Tree Traversal without recursion and without stack (Threaded binary tree)-

Threaded Binary Tree-




```
struct node {
    int data;
    struct node * lchild;
    struct node * right;
    int rthreadPtr;
}
```

2- Threaded Binary Tree-



lthreadPtr- Inorder predecessor

rthreadPtr- Inorder successor

Approach- Use threaded Binary Tree

T(n) = O(n)

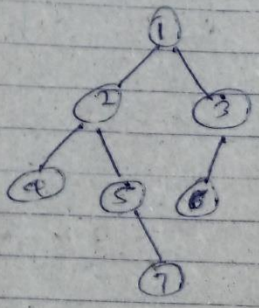
S(n) = O(1)

Algo-

```
p = leftmostnode(root)
while(p != NULL)
{
    print p->data;
    if p->rthreadPtr
        p = p->right;
    else
        p = leftmostnode(p->right);
}
```

(11)

5.18 Serialize and deserialize of a binary tree



Approach - We use preorder for store in hard disk.

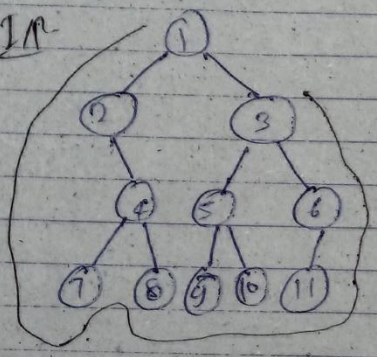
Internal Node - [1|1]

Leaf Node - [4|0] → Internal Node

→ External Node

5.19 Boundary Traversal of a Binary Tree

I/P

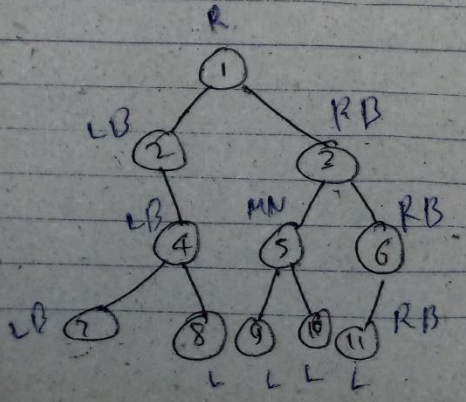


O/P - 1, 2, 4, 7, 8, 9, 10, 11, 6, 3

Approach :- Traverse in Preorder + Apply Cases

Print R, LBs, L's, RB's

(order) (order) (order)

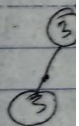


R, LB, RB, L, MN

Auxillary info

5.20

I/P



Appr

Code

(113)

Case 1: Root ← LC → LB
RC → RB

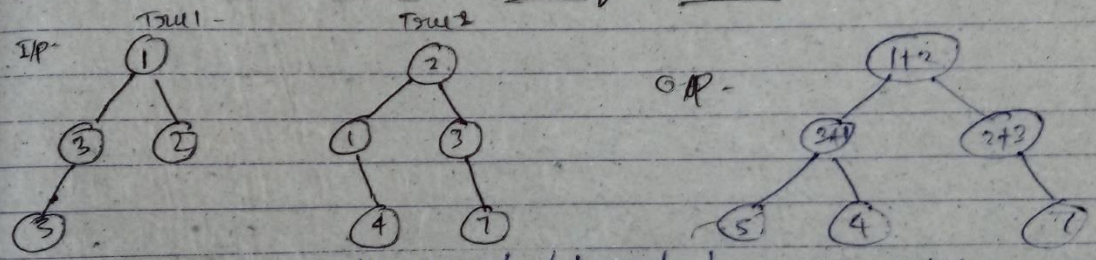
Case 2: LB ← LC → LB
Re → LB if LC is NULL

Case 3: RB ← LC → RB if RC is NULL
RC → RB

Case 4: leaves

Case 5: NN

5.20 Merge Two binary Trees



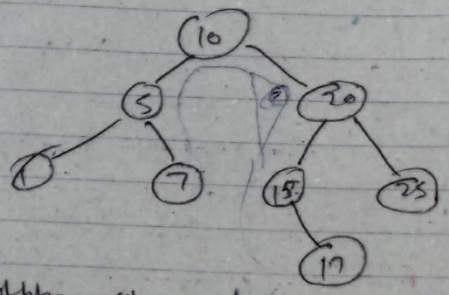
Approach- use traversal (preorder) T(N) = O(n) S(n) = O(h)

Code-
```
TreeNode * mergeTrees(TreeNode* root1, TreeNode* root2)
{
    if(!root1 && !root2)
        return nullptr;
    if(!root1)
        return root2;
    if(!root2)
        return root1;
    TreeNode * temp = new TreeNode(root1->val+root2->val);
    temp->left = mergeTrees(root1->left, root2->left);
    temp->right = mergeTrees(root1->right, root2->right);
    return temp;
}
```

(113)

5.21 Range Sum of Binary search Tree -

I/P - L = 7, R = 17



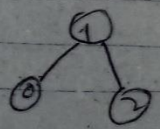
O/P = 32

Approach - Use traversal. T(n) = O(n) S(n) = O(n)

Code - int rangeSumBST (TreeNode* root, int low, int high)
{
if(!root)
return 0;
if(low <= root->val && root->val <= high)
return root->val + rangeSumBST(root->left, low, high) + rangeSumBST(root->right, low, high);
else
return rangeSumBST(root->left, low, high) + rangeSumBST(root->right, low, high);
}

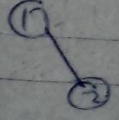
5.22 - Trim a Binary Search Tree -

I/P



L = 1, R = 2

O/P -



(114)

Approach - Use traversal. T(n) = O(n) Sp = O(h)

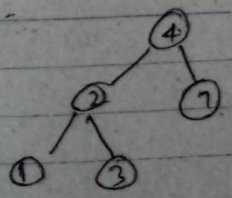
Code -

```
TreeNode* trimBST(TreeNode* root, int low, int high)
{
    if(!root)
        return nullptr;
    if(root->val < low)
        return trimBST(root->right, low, high);
    else if(root->val > high)
        return trimBST(root->left, low, high);
    root->left = trimBST(root->left, low, high);
    root->right = trimBST(root->right, low, high);
    return root;
}
```

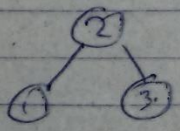
5.13 Search in a Binary Tree -

Problem Statement - Given the root node of a binary search tree (BST) and a value. You need to find the node in the BST. that the node's value equals the given value. Return the subtree rooted with that node. If such node doesn't exist, you should return NULL.

I/p -



O/p -



Search: 2.

115

Approach:- Recursive Approach - T(n) = O(h) Sm = O(h)

Code-

```
TreeNode * searchBST (TreeNode * root, int val)
{
    if (!root || root->val == val)
        return root;
    if (root->val > val)
        return searchBST (root->left, val);
    return searchBST (root->right, val);
}
```

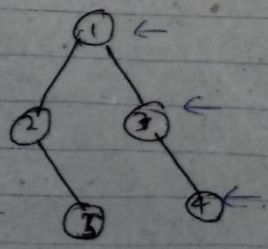
Approach2- Iterative Approach: T(n) = O(h) Sm = O(1)

Code-

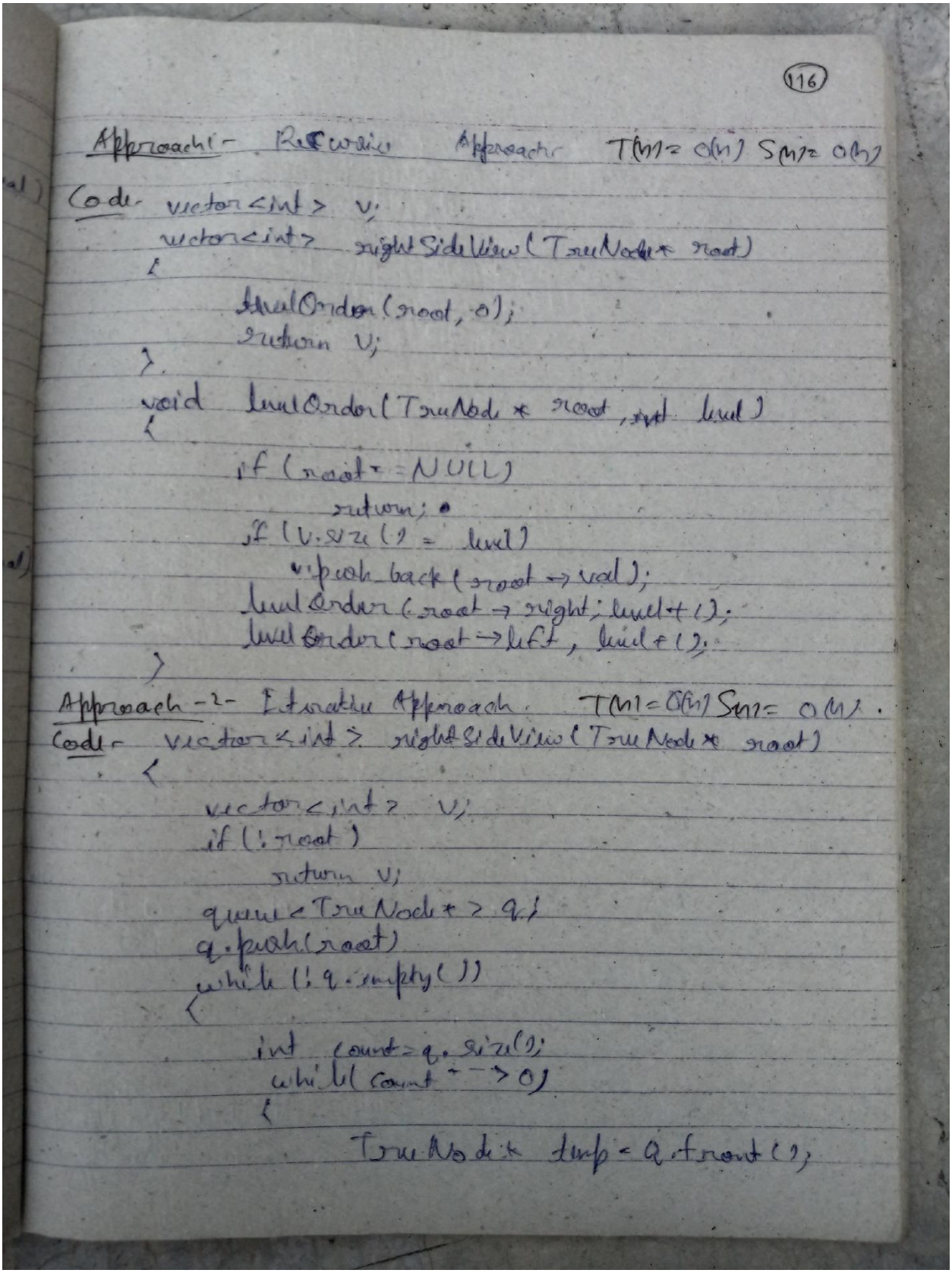
```
TreeNode * searchBST(TreeNode * root, int val)
{
    while (root != NULL && root->val != val)
        if (root->val > val)
            root = root->left;
        else
            root = root->right;
    return root;
}
```

5.2 Print Right View of a Binary Tree-

Ip-



⇒ Op- 1, 3, 4



(116)

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

Code:
```
vector<int> v;
vector<int> rightSideView(TreeNode* root)
{
    levelOrder(root, 0);
    return v;
}
void levelOrder(TreeNode* root, int level)
{
    if (root == NULL)
        return;
    if (v.size() == level)
        v.push_back(root->val);
    levelOrder(root->right, level+1);
    levelOrder(root->left, level+1);
}
```

Approach-2:- Iterative Approach. T(n) = O(n) S(n) = O(n).

Code:
```
vector<int> rightSideView(TreeNode* root)
{
    vector<int> v;
    if (!root)
        return v;
    queue<TreeNode*> q;
    q.push(root)
    while (!q.empty())
    {
        int count = q.size();
        while(count-- > 0)
        {
            TreeNode* temp = q.front();
```

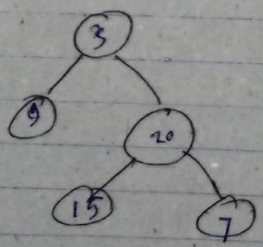
(117)

```
        q.pop();
        if(count==0)
            v.push_back(temp->val);
        if(temp->left!=NULL)
            q.push(temp->left);
        if(temp->right!=NULL)
            q.push(temp->right);
    }
}
return V;
}
```

5.26 Maximum Depth of Binary Tree -

Problem Statement - The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

I/P



O/P 3

Approach Use Recursion TM = O(h) SM = O(h)

Code

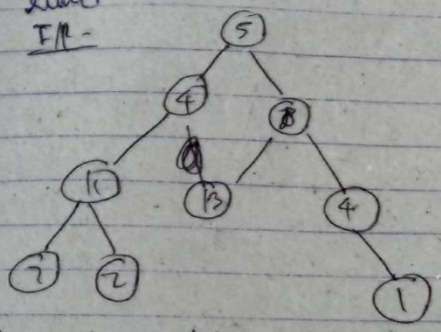
```
int maxDepth(TreeNode* root)
{
    if(!root)
        root 0;
    return 1+max(maxDepth(root->left),maxDepth(root->right));
}
```

118

5.27 Path Sum-

Problem Statement- Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

I/P-



O/P- true

(5 → 4 → 11 → 2) when sum is 22

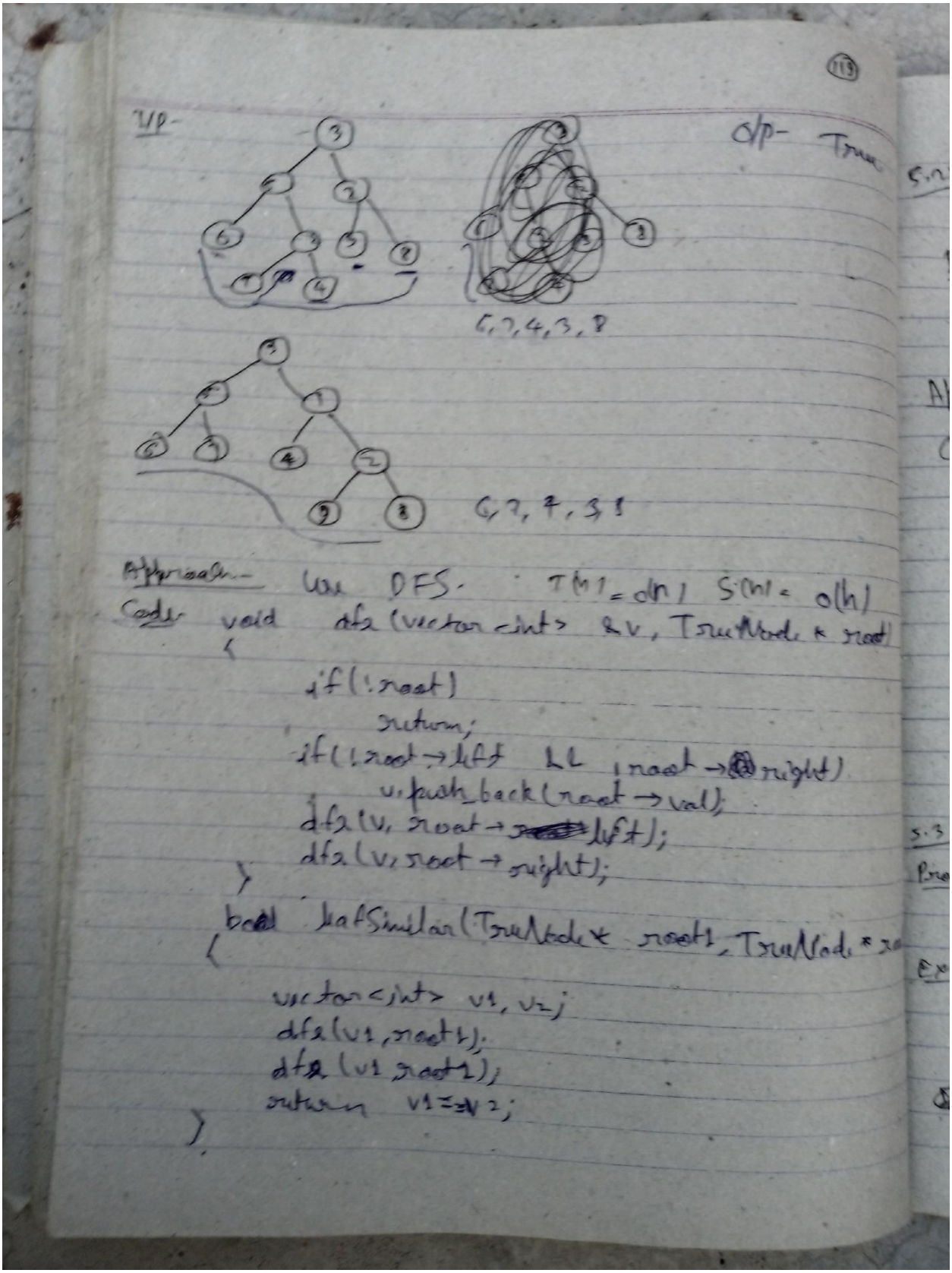
Approach- Use recursion T(n) = O(n) S(n) = O(h)

Code-

```
bool hasPathSum(TreeNode * root, int targetSum)
{
    if (!root)
        return false;
    targetSum = root->val;
    if(!root->left && !root->right)
        return targetSum == 0;
    return hasPathSum(root->left, targetSum) ||
           hasPathSum(root->right, targetSum);
}
```

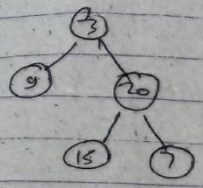
5.28 Leaf-Similar Trees-

Problem Statement- Consider all the leaves of a binary tree, from left to right order, the values of those leaves form a leaf value sequence.


(19)
I/p-
O/p- True
6, 7, 4, 3, 8
6, 7, 4, 3, 8
Approach- Use DFS. TM = O(n) SM = O(h)
Code- void dfs (vector<int> &v, TreeNode * root)
{
if(!root)
return;
if(!root->left && root->right)
v.push_back(root->val);
dfs(v, root->left);
dfs(v, root->right);
}
bool leafSimilar(TreeNode* root1, TreeNode* ro
{
vector<int> v1, v2;
dfs(v1, root1);
dfs(v1, root2);
return v1==v2;
}

120

5.29 Find the sum of all left leaves in a given binary tree.

Ex- 

O/P - 9 + 15 = 24

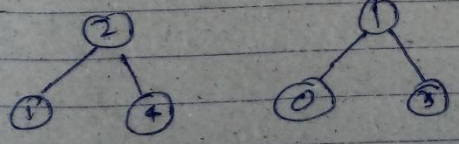
Approach - Use Recursion. T(n) = O(n) S(n) = O(h)

Code-

```
int sumOfLeftLeaves(TreeNode* root)
{
    int result = 0;
    if(!root)
        return result;
    if(root->left && !root->left->left
            && !root->left->right)
        result += root->left->val;
    return result + sumOfLeftLeaves(root->left)
            + sumOfLeaves(root->right);
}
```

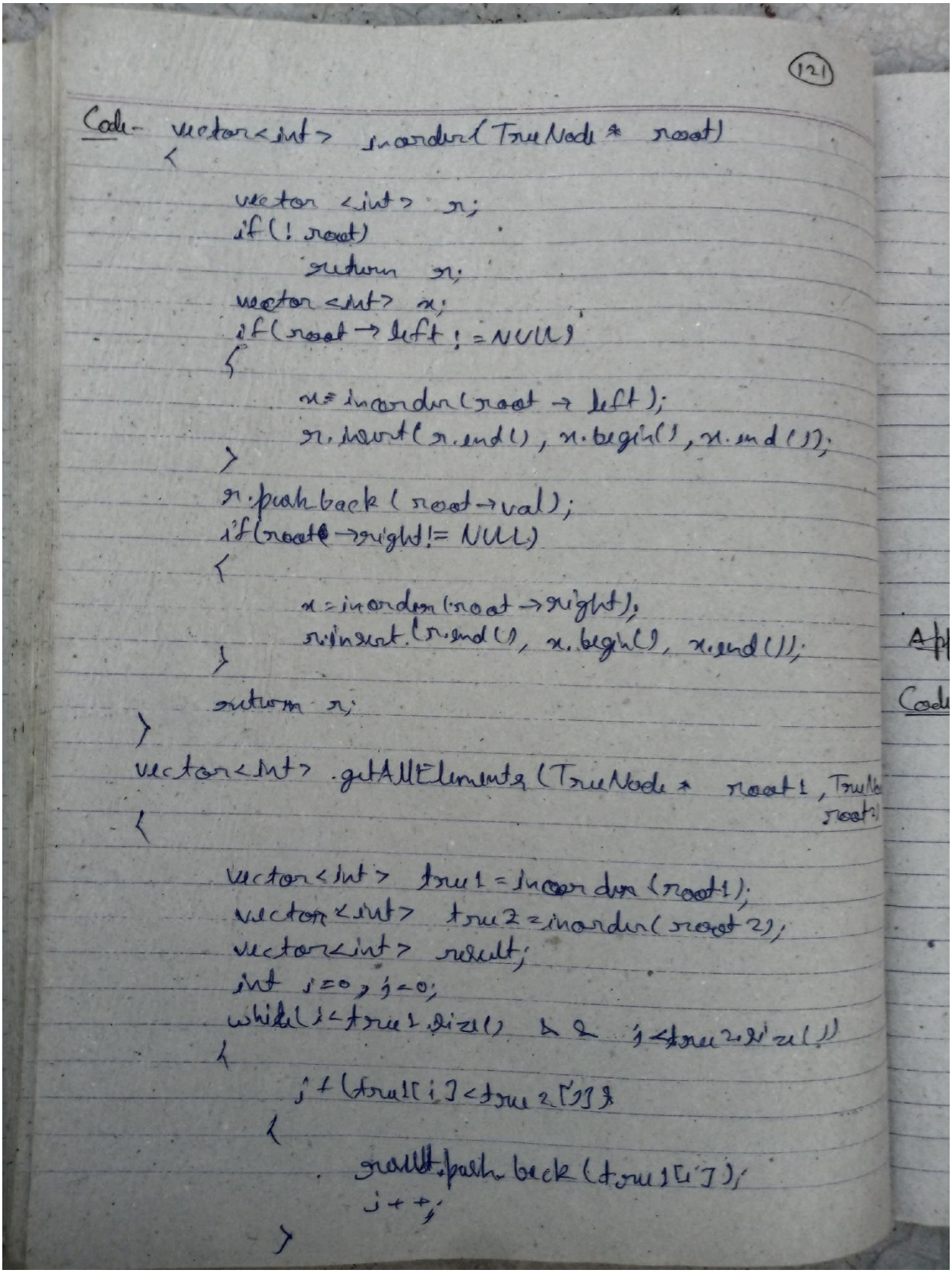
5.31 All Elements in Two Binary Search Trees -

Problem Statement - Given two binary trees root1 and root2. Return a list containing all the integers from both trees. Stored in ascending order.

Ex- 

O/P - [0, 1, 1, 2, 3, 4]

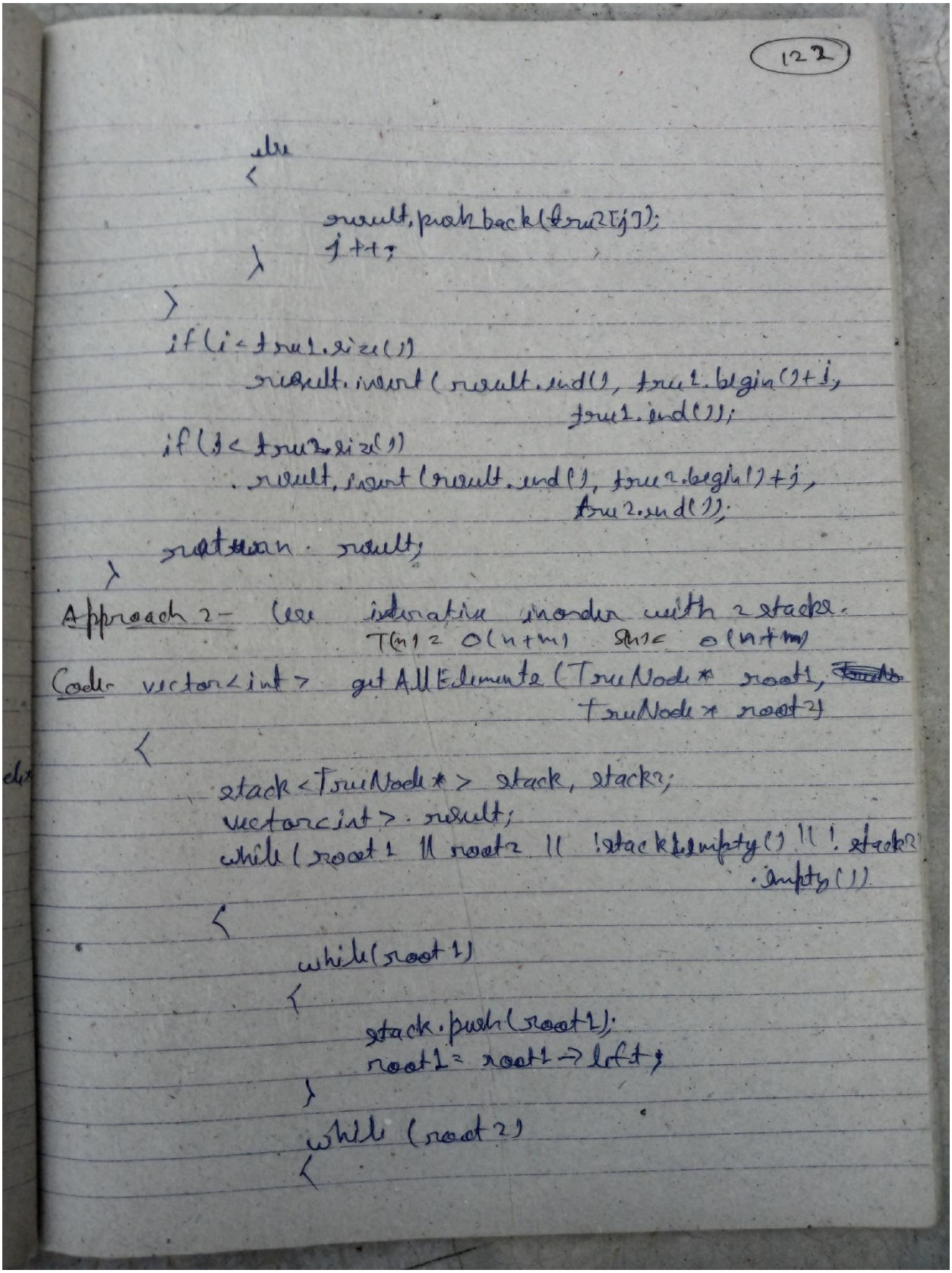
Approach :- Use inorder traversal and merge algorithm. T(n) = O(m+n) S(n) = O(m+n)



(121)

Code-

```
vector<int> inorder(TreeNode* root)
{
    vector<int> x;
    if(!root)
        return x;
    vector<int> x;
    if(root->left != NULL)
    {
        x = inorder(root->left);
        x.insert(x.end(), x.begin(), x.end());
    }
    x.push_back(root->val);
    if(root->right != NULL)
    {
        x = inorder(root->right);
        x.insert(x.end(), x.begin(), x.end());
    }
    return x;
}
vector<int> getAllElements(TreeNode* root1, TreeNode* root2)
{
    vector<int> tree1 = inorder(root1);
    vector<int> tree2 = inorder(root2);
    vector<int> result;
    int i=0, j=0;
    while(i<tree1.size() && j<tree2.size())
    {
        if(tree1[i]<tree2[j])
        {
            result.push_back(tree1[i]);
            i++;
        }
```

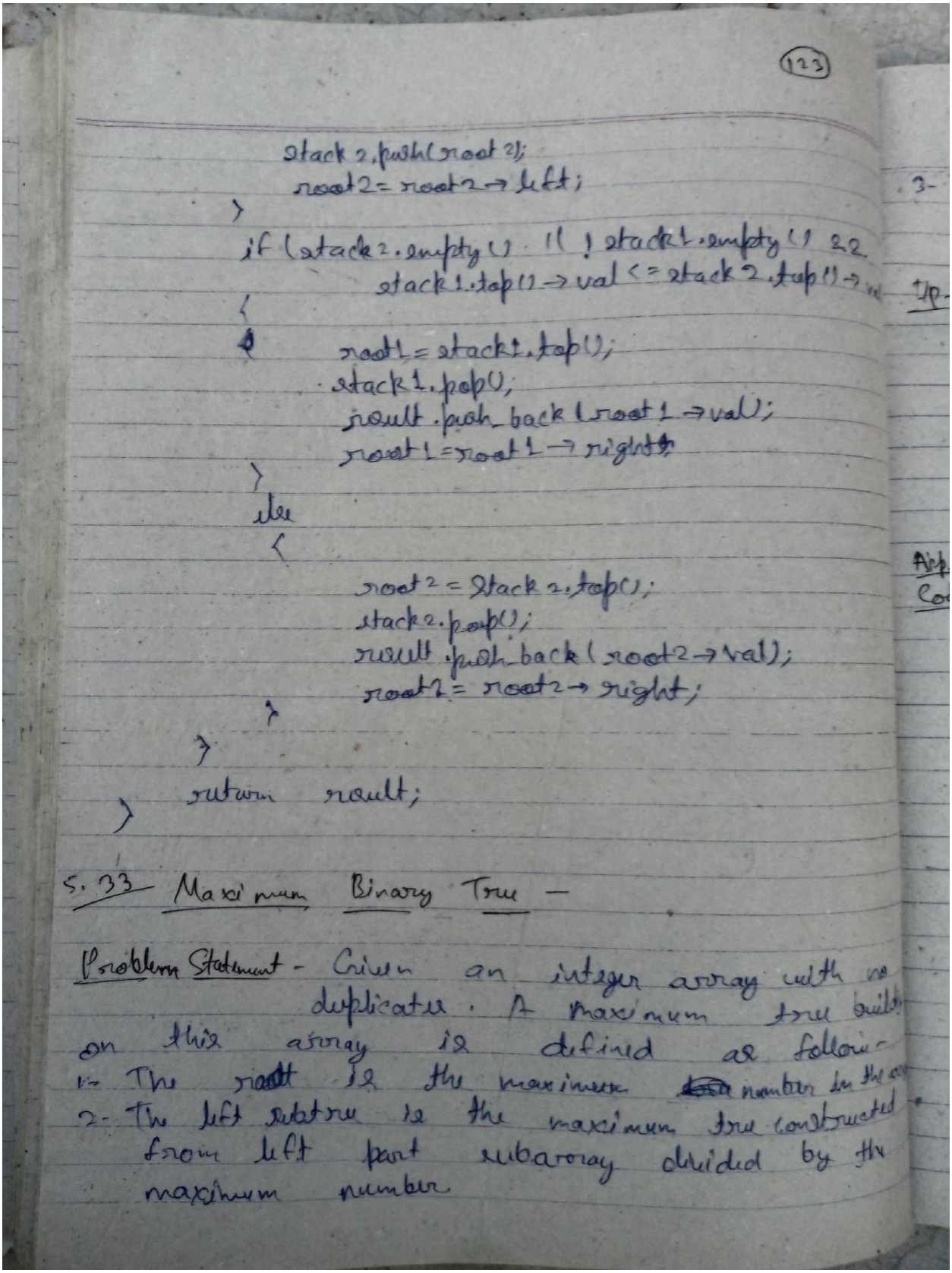


122

```
        else
        {
            result.push_back(true2[j]);
            j++;
        }
    }
    if(i<true1.size())
        result.insert(result.end(), true1.begin()+i,
                                    true1.end());
    if(j<true2.size())
        result.insert(result.end(), true2.begin()+j,
                                    true2.end());
    return result;
}
```

Approach 2 - (Iterative inorder with 2 stacks.
T(n) = O(n+m) Sp = O(n+m)

```
Code: vector<int> getAllElements(TrueNode* root1,
                                 TrueNode* root2)
{
    stack<TrueNode*> stack1, stack2;
    vector<int> result;
    while(root1 || root2 || !stack1.empty() || !stack2
                                 .empty())
    {
        while(root1)
        {
            stack1.push(root1);
            root1 = root1->left;
        }
        while(root2)
        {
```



```
        stack2.push(root2);
        root2 = root2->left;
    }
    if (stack2.empty() || !stack1.empty() &&
            stack1.top()->val <= stack2.top()->val)
    {
        root1 = stack1.top();
        stack1.pop();
        result.push_back(root1->val);
        root1 = root1->right;
    }
    else
    {
        root2 = stack2.top();
        stack2.pop();
        result.push_back(root2->val);
        root2 = root2->right;
    }
}
return result;
}
```

5.33 Maximum Binary Tree –

Problem Statement - Given an integer array with no duplicates. A maximum tree built on this array is defined as follow:-

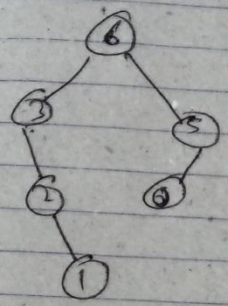
1- The root is the maximum number in the array.

2- The left subtree is the maximum tree constructed from left part subarray divided by the maximum number.

(24)

3- The right subtree is the maximum tree constructed from right part. subarray divided by the maximum number.

I/p- [2, 7, 1, 5, 0, 5]



Approach- We Recursion T(n) = O(n log n) S(n) = O(n).

Code-

```
TreeNode* constructMaximumBinaryTree(vector<int>& nums)
{
    if (nums.empty())
        return nullptr;
    auto it = max_element(nums.begin(), nums.end());
    TreeNode* root = new TreeNode(*it);
    vector<int> left(nums.begin(), it);
    root->left = constructMaximumBinaryTree(left);
    vector<int> right(++it, nums.end());
    root->right = constructMaximumBinaryTree(right);
    return root;
}
```

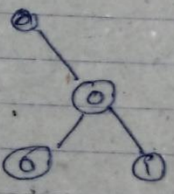
125

5.36 Binary Tree Pruning –

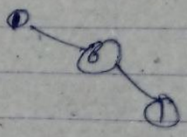
Problem Statement – We are given the head node root of a binary tree, where additionally every node's value is either a 0 or a 1.

Return the same tree where every subtree (of the given tree) not containing a 1 has been removed.

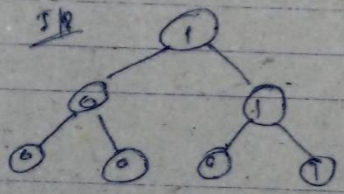
I/P



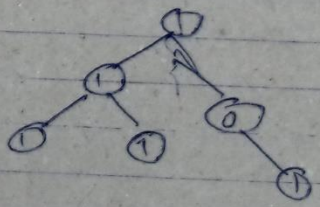
O/P –



I/P



O/P



Approach :- Recursion $T(n) = O(n)$ $S(n) = O(h)$

Code –

```
TreeNode* pruneTree(TreeNode* root)
{
    if(!root)
        return nullptr;
    root->left = pruneTree(root->left);
    root->right = pruneTree(root->right);
    if(!root->left && !root->right &&
                        root->val == 0)
        return nullptr;
    return root;
}
```

(126)

5.38 Validate Binary Search Tree-

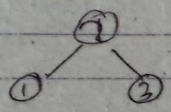
Problem Statement- Given the root of a binary tree, determine if it is a valid binary search tree (BST).

A valid BST is defined as follows-

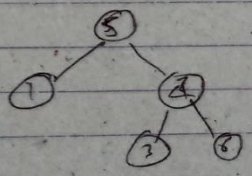
(i) The left subtree of a node contains only nodes with keys less than the node's key.

(ii) The right subtree of a node contains only nodes with keys greater than the node's key.

(iii) Both the left and right subtrees must also be binary search trees.

I/P- 

O/P- true.

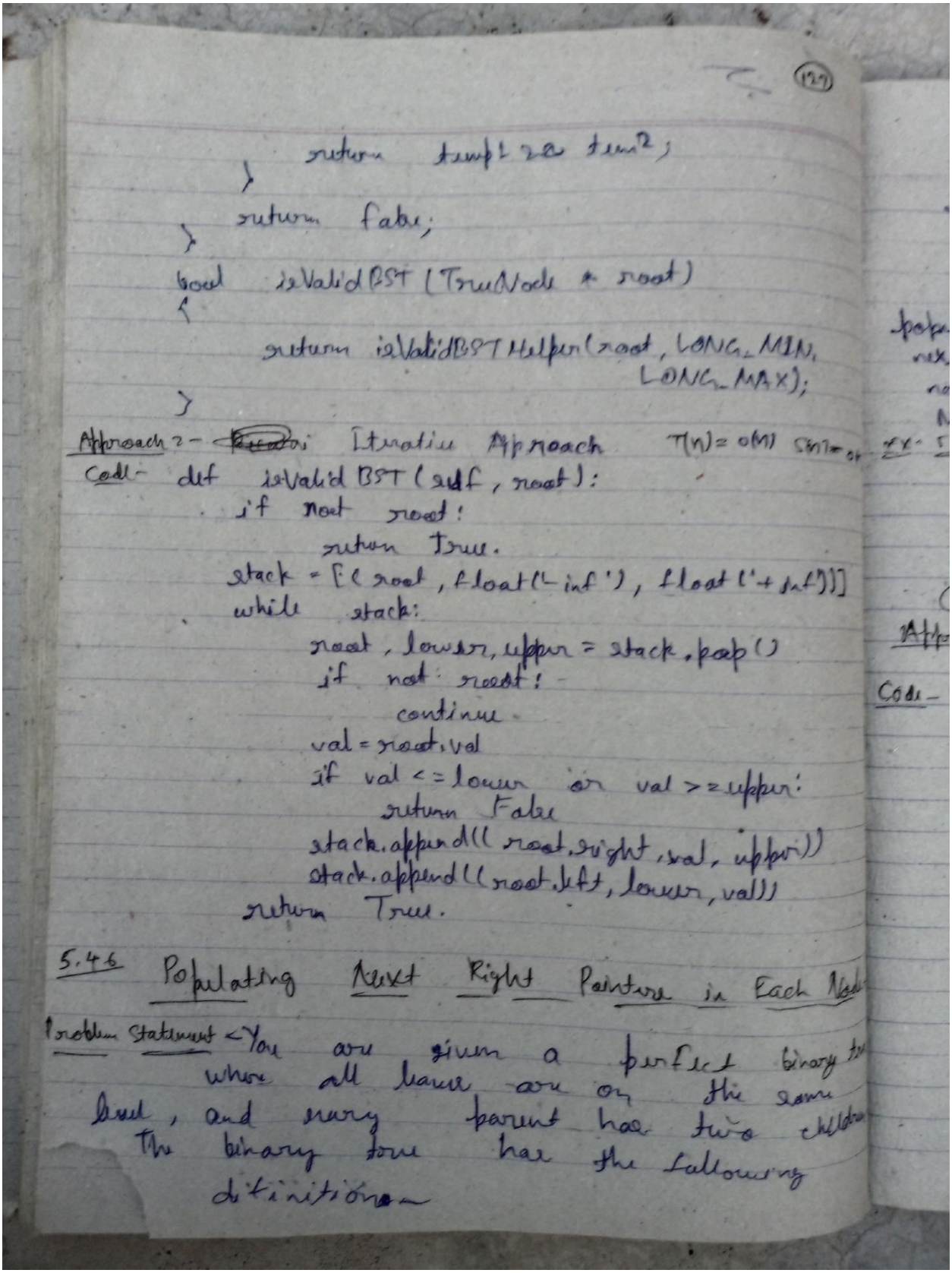
I/P- 

O/P- False

Approach I- Recursive Approach- T(n) = O(n) S(n) = O(n)

Code-

```
bool isValidBSTHelper(TreeNode * root, long min,
                      long max)
{
    if(!root)
        return true;
    if(root->val > min && root->val < max)
    {
        bool temp1 = isValidBSTHelper(root->left, min,
                                      root->val);
        bool temp2 = isValidBSTHelper(root->right,
                                      root->val, max);
```



```
        return temp1 && temp2;
    }
    return false;
}
bool isValidBST(TreeNode* root)
{
    return isValidBSTHelper(root, LONG_MIN,
                            LONG_MAX);
}
```

Approach 2 - Iterative Approach T(N) = O(N) S(N) = O(N)

Code -

```
def isValidBST(self, root):
    if not root:
        return True.
    stack = [(root, float('-inf'), float('+inf'))]
    while stack:
        root, lower, upper = stack.pop()
        if not root:
            continue
        val = root.val
        if val <= lower or val >= upper:
            return False
        stack.append((root.right, val, upper))
        stack.append((root.left, lower, val))
    return True.
```

5.46 Populating Next Right Pointers in Each Node

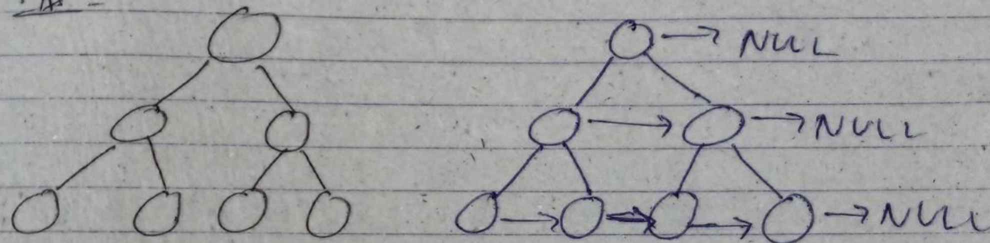
Problem Statement - You are given a perfect binary tree where all leaves are on the same level, and every parent has two children. The binary tree has the following definition -

(128)

```
Struct Node {
    int val
    Node *left, *right, *next;
}
```

populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Ex- 5A -



Approach - Use DFS (Level order traversal)
T(n) = O(n)   S(n) = O(n + k).   where k = No. of level

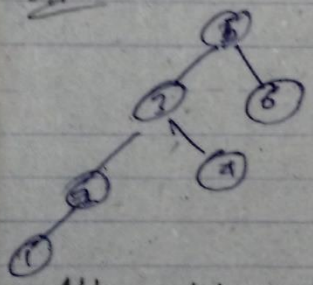
Code -
```
Node * connect(Node * root)
{
    if(!root)
        return root;
    queue<Node *> q;
    q.push(root);
    q.push(NULL);
    while(!q.empty())
    {
        Node * current = q.front();
        q.pop();
        if(current==NULL)
        {
            if(!q.empty())
                q.push(NULL);
        }
```

```
else
{
    current -> next = q.front();
    if(current -> left)
        q.push(current -> left);
    if(current -> right)
        q.push(current -> right);
}
}
return root;
}
```

5.52 Kth Smallest Element in a BST.

Understand - Given root of a binary search tree and an integer K, return the k^{th} (1-indexed) smallest element in the tree.

EX- 3

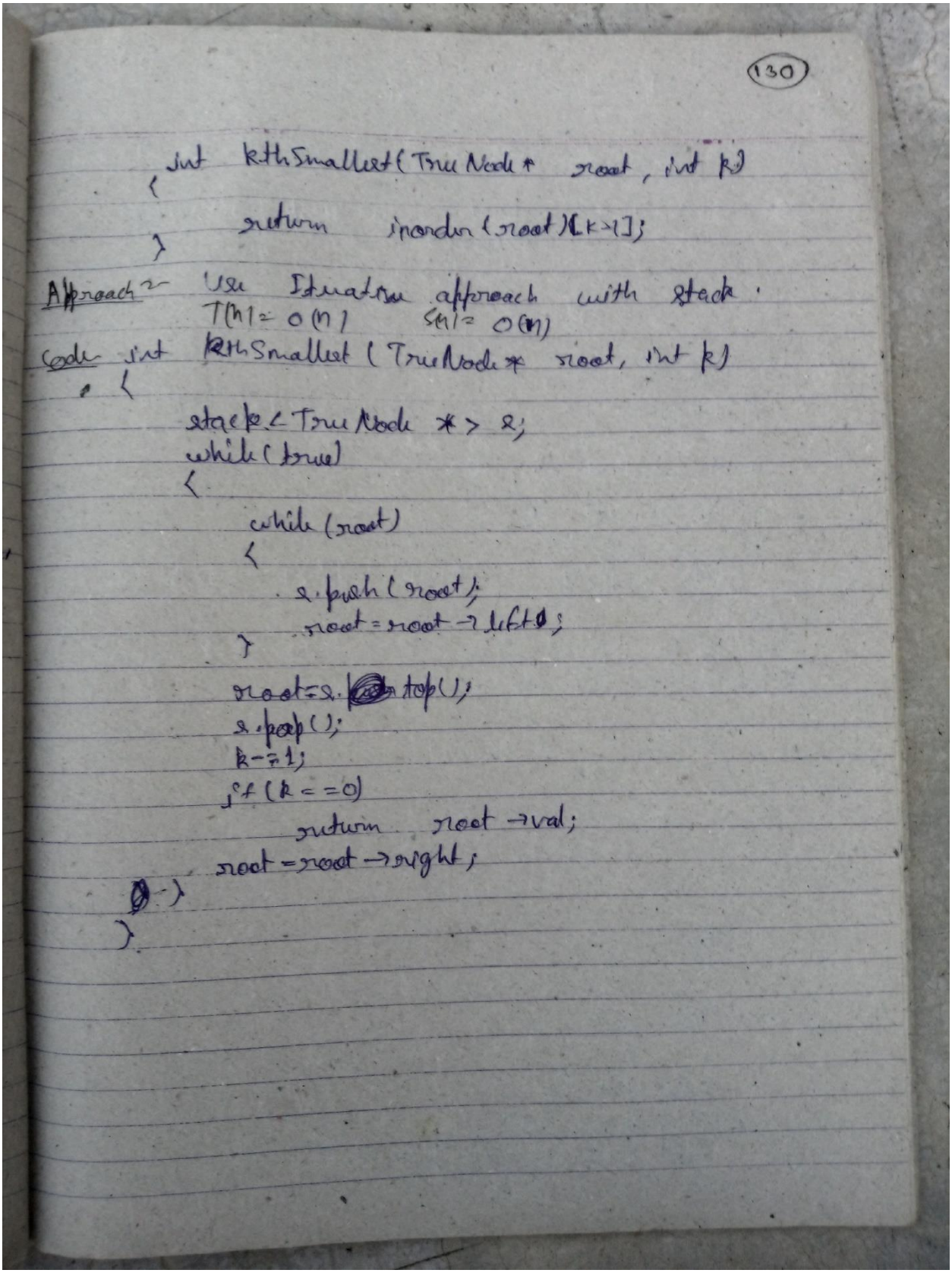


OR 3

Approach - Use inorder traversal with extra space $S(n) = O(n)$ $T(n) = O(n)$.

Code -

```
vector<int> v;
vector<int> inorder(TreeNode* root)
{
    if(!root)
        return v;
    inorder(root -> left);
    v.push_back(root -> val);
    inorder(root -> right);
    return v;
}
```

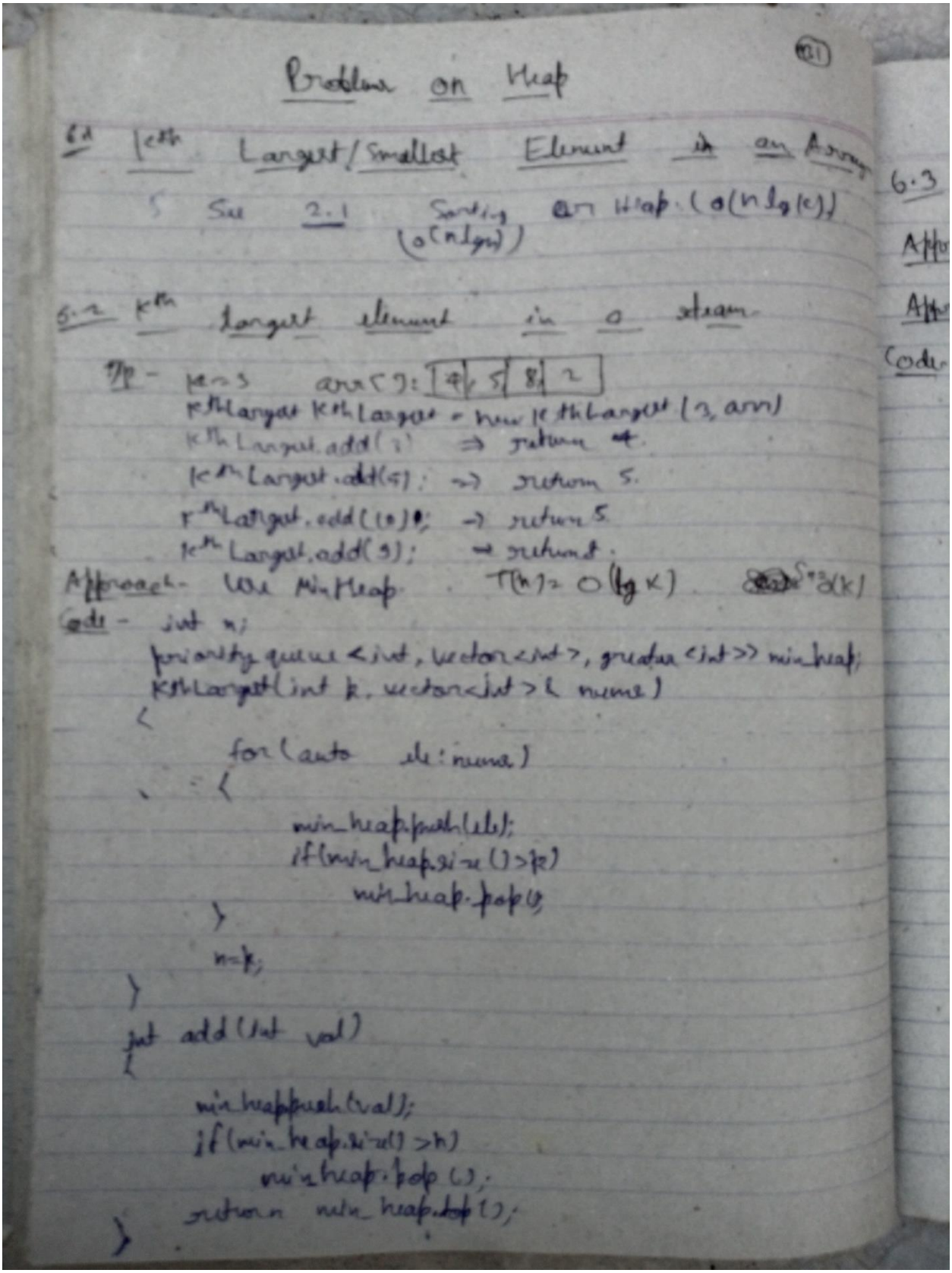


130

```
int kthSmallest(TreeNode* root, int k)
{
    return inorder(root)[k-1];
}
```

Approach :- Use Iterative approach with stack.
T(n) = O(n) S(n) = O(n)

Code
```
int kthSmallest(TreeNode* root, int k)
{
    stack<TreeNode*> s;
    while(true)
    {
        while(root)
        {
            s.push(root);
            root = root->left;
        }
        root = s.top();
        s.pop();
        k--;
        if(k==0)
            return root->val;
        root = root->right;
    }
}
```



Problem on Heap

6.1 k^{th} Largest/Smallest Element in an Array

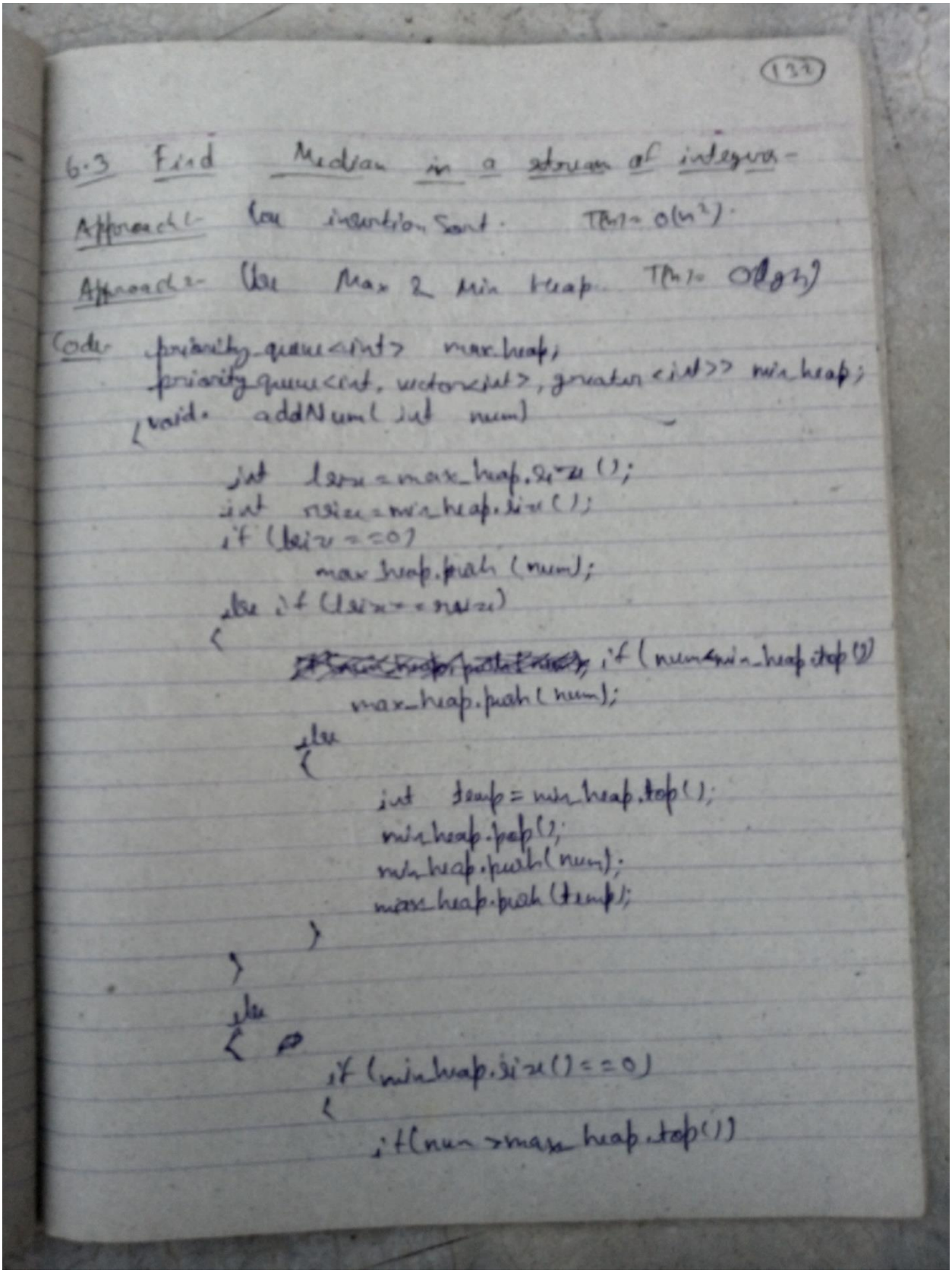
See 2.1 Sorting on Heap. ($O(n \log k)$)
($O(n \log n)$)

6.2 K^{th} largest element in a stream

Ex - k=3 arr[]: [4, 5, 8, 2]
KthLargest KthLargest = new KthLargest(3, arr)
KthLargest.add(3) ⇒ return 4.
K^{th}Largest.add(5); → return 5.
K^{th}Largest.add(10); → return 5
K^{th}Largest.add(9); → return 8.
Approach - Use MinHeap. $T(n) = O(\log K)$ $S = O(K)$
Code -

```
int n;
priority_queue<int, vector<int>, greater<int>> min_heap;
KthLargest(int k, vector<int>& nums)
{
    for(auto ele:nums)
    {
        min_heap.push(ele);
        if(min_heap.size()>k)
            min_heap.pop();
    }
    n=k;
}
int add(int val)
{
    min_heap.push(val);
    if(min_heap.size()>n)
        min_heap.pop();
    return min_heap.top();
}
```

6.3
Appr
Appr
Code



(132)

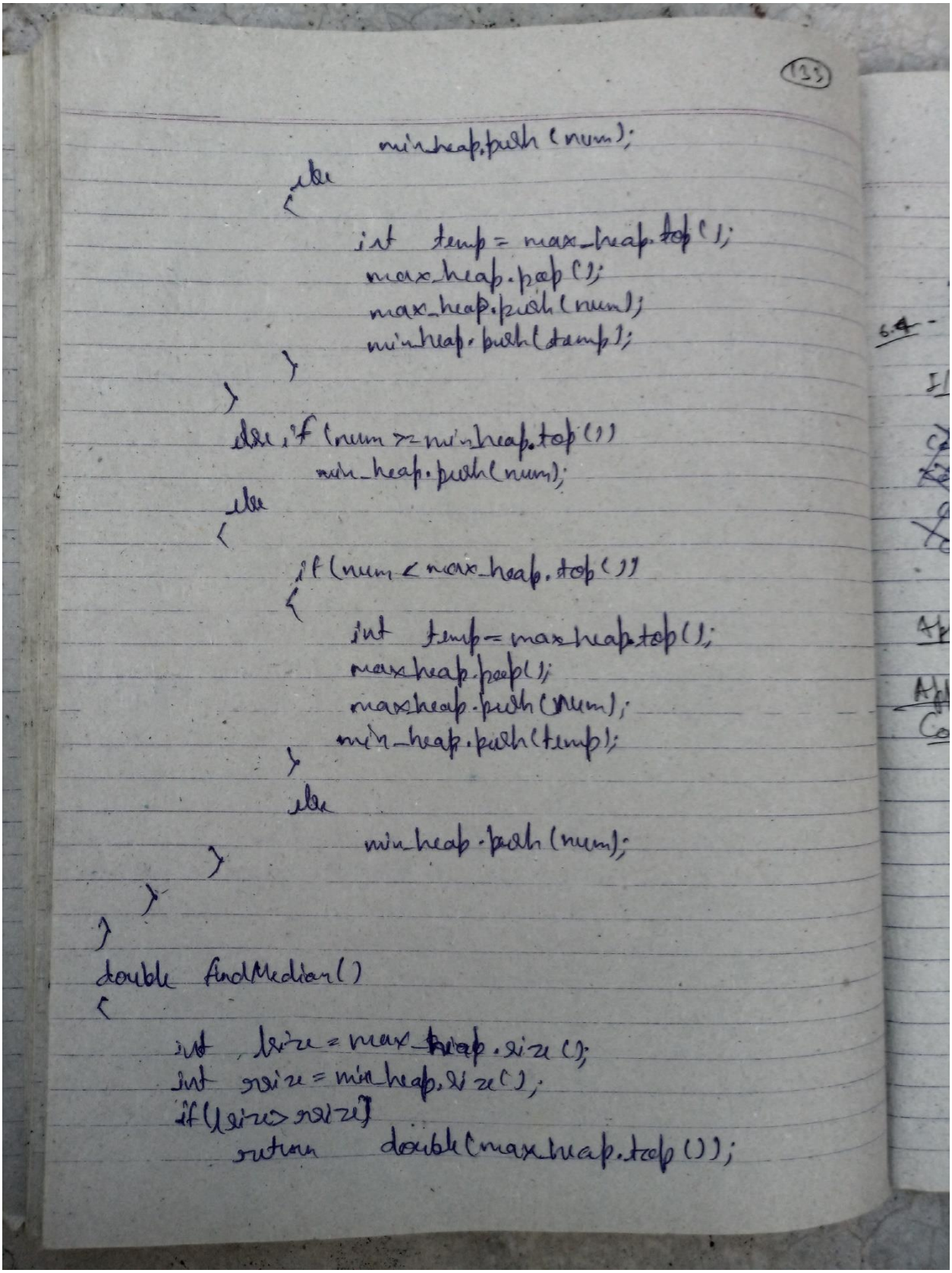
6.3 Find Median in a stream of integers -

Approach 1- Use insertion Sort. TC = $O(n^2)$.

Approach 2- Use Max & Min Heap. TC = $O(\log n)$

Code

```
priority_queue<int> max_heap;
priority_queue<int, vector<int>, greater<int>> min_heap;
void addNum(int num)
{
    int lsize = max_heap.size();
    int rsize = min_heap.size();
    if (lsize == 0)
        max_heap.push(num);
    else if (lsize == rsize)
    {
        if (num < min_heap.top())
            max_heap.push(num);
        else
        {
            int temp = min_heap.top();
            min_heap.pop();
            min_heap.push(num);
            max_heap.push(temp);
        }
    }
    else
    {
        if (min_heap.size() == 0)
        {
            if (num > max_heap.top())
```



```
                min_heap.push(num);
            else
            {
                int temp = max_heap.top();
                max_heap.pop();
                max_heap.push(num);
                min_heap.push(temp);
            }
        }
        else if (num >= min_heap.top())
            min_heap.push(num);
        else
        {
            if(num < max_heap.top())
            {
                int temp = max_heap.top();
                max_heap.pop();
                max_heap.push(num);
                min_heap.push(temp);
            }
            else
                min_heap.push(num);
        }
    }
}
double findMedian()
{
    int size = max_heap.size();
    int rsize = min_heap.size();
    if(size > rsize)
        return double(max_heap.top());
```

(34)

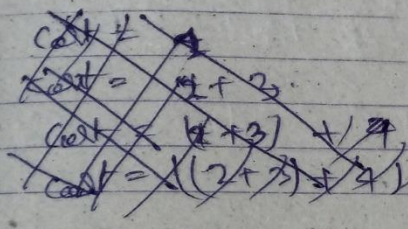
else

return (double(max_heap.top())+ double(min_heap.top()))/2;

}

6.4 - Connect n ropes with minimum cost -

I/P - n=4 arr[]: {4, 3, 2, 6} op = 29.



2+3 = 5 | arr={4, 5, 6}
4+5 = 9 | arr={6, 9}
9+6 = 15. arr[15]
+
29

Approach-1 - use sorting T(n) = O(n^3)

Approach-2 - Use Min Heap. T(n) = O(n log n)

Code - long long minCost(long long arr[], long long n)

{

priority_queue<long long, vector<long long>, greater<long long>> min_heap(arr, arr+n);

long long cost = 0;

while(min_heap.size()>1)

{

long long first = min_heap.top();

minheap.pop();

long long second = min_heap.top();

min_heap.pop();

cost += (first + second);

135

```
        min_heap.push(first + second);
    }
    return cost;
}
```

Code

6.5 Convert Min Heap to Max Heap -

Use MaxHeap() function of Heap sort which use intrally Heapify() method. T(n) = O(n)

6.6- Finding K- Most frequent words in a text file -

w1 w2 w1 w3 w4 w- w3 w5 ——.

K=3

n words.
n - unique

use HashMap to store frequency of words

word	Count
w1	3
w2	10
w3	1
w4	6
k	k

T(n) = O(n).

hash table unsorted.

S(n) = O(n)

1) Read words - T(n) = O(n)

2) Count the frequency of each word - T(n) = O(n), S(n) = O(n).

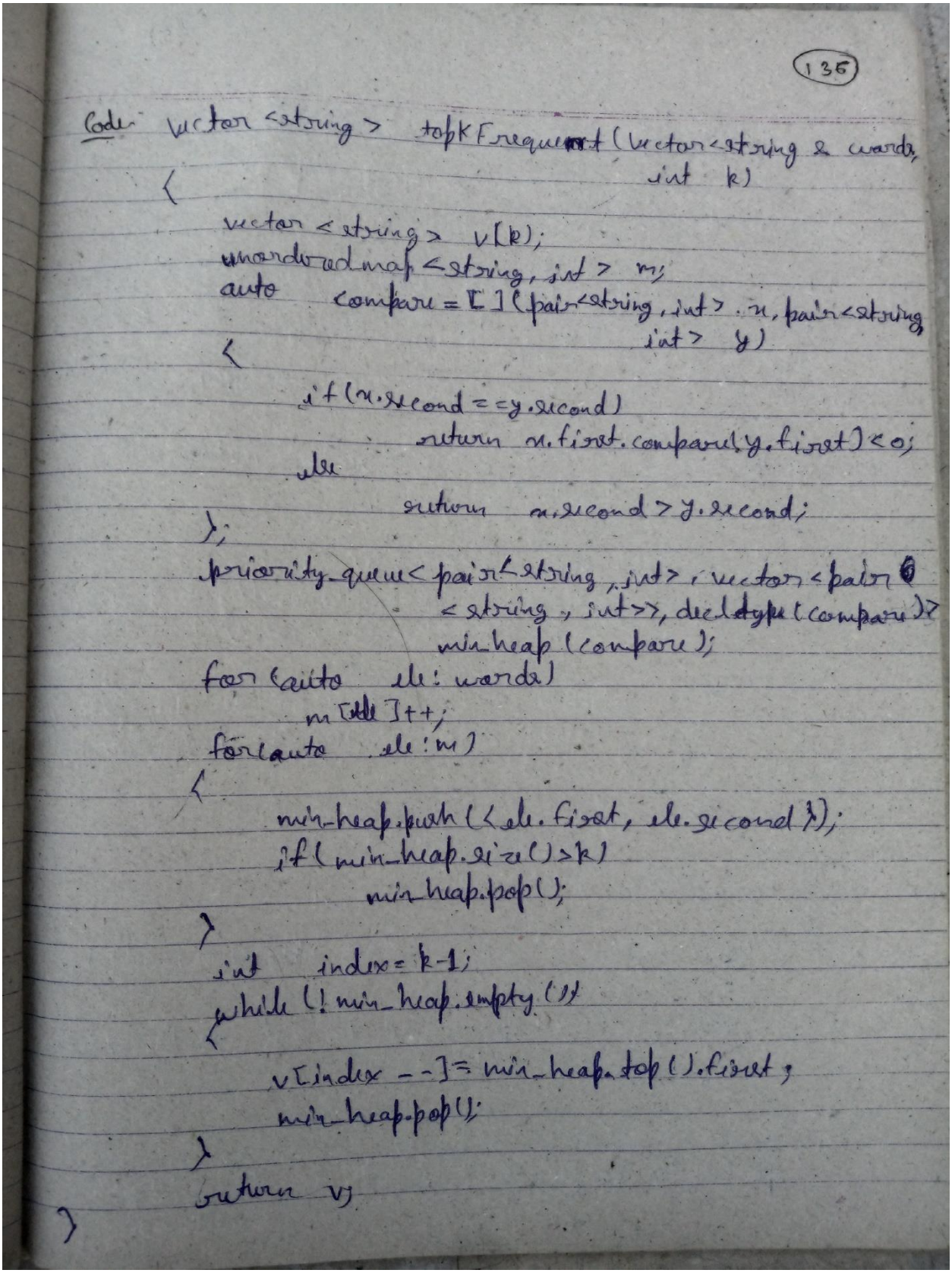
3) Find K- most frequent words

(a) If use sorting - T(n) = O(n log n) + O(k).

(b) If use MaxHeap T = O(n) + O(n) + O(k log n)

S(n) = O(n)

}



135

Code:

```
vector<string> topKFrequent(vector<string>& words,
                            int k)
{
    vector<string> v(k);
    unordered_map<string, int> m;
    auto compare = [](pair<string, int> x, pair<string,
                      int> y)
    {
        if(x.second == y.second)
            return x.first.compare(y.first) < 0;
        else
            return x.second > y.second;
    };
    priority_queue<pair<string, int>, vector<pair
        <string, int>>, decltype(compare)>
        min_heap(compare);
    for(auto ele: words)
        m[ele]++;
    for(auto ele: m)
    {
        min_heap.push({ele.first, ele.second});
        if(min_heap.size() > k)
            min_heap.pop();
    }
    int index = k-1;
    while(!min_heap.empty())
    {
        v[index--] = min_heap.top().first;
        min_heap.pop();
    }
    return v;
}
```

137

6.7 K closest points to origin -

Ip- : [[3, 3], [5, -1], [-2, 4]] K=2

op- [[3, 3], [-2, 4]].

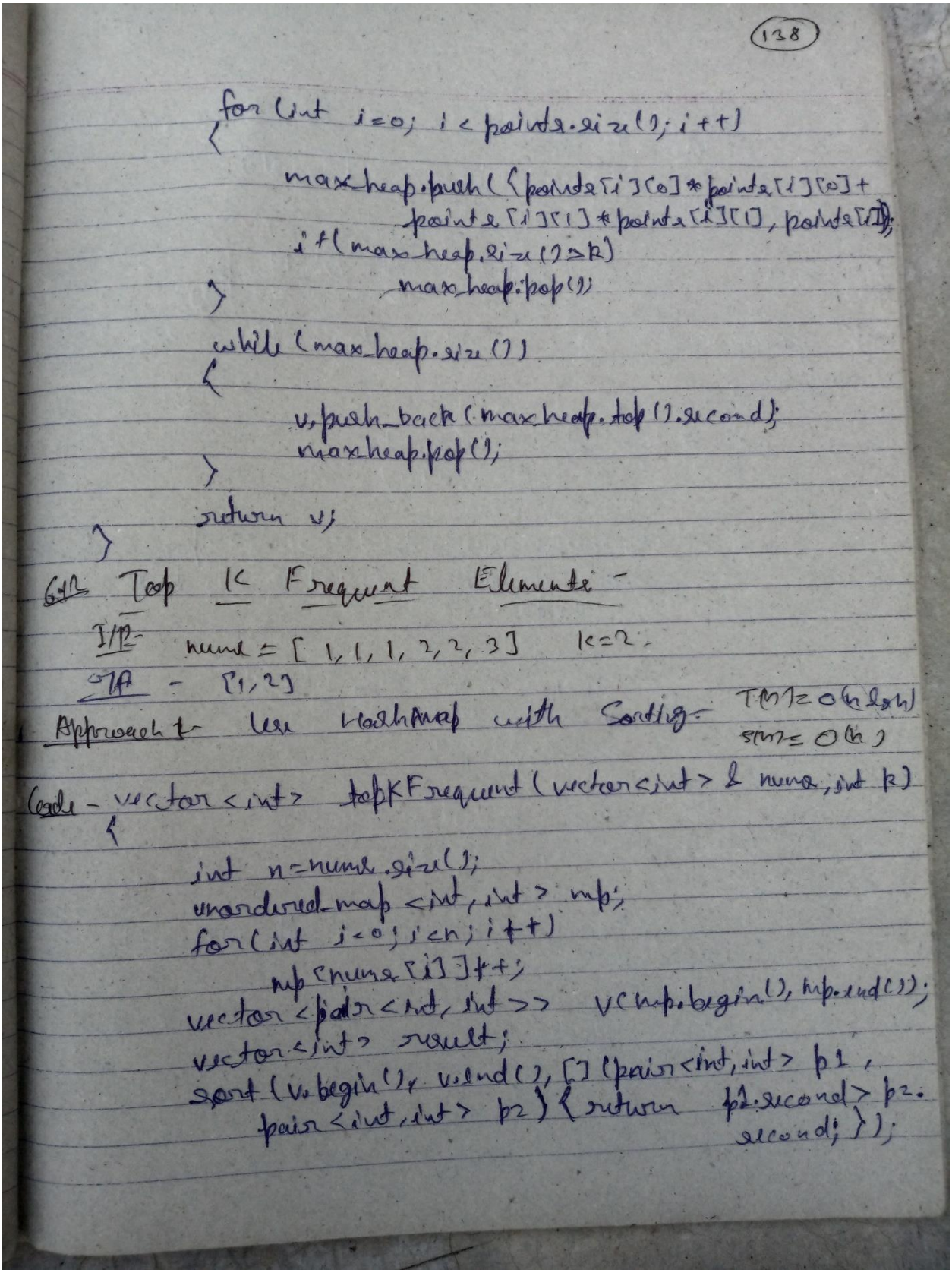
We use $d = \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}$ formula to check a point how closer to the origin.

Approach1- We Sorting- T(n) = O(nlogn) S(n) = O(1)

Code-
```
vector<vector<int>> kClosest(vector<vector<int>>& points, int k)
{
    vector<vector<int>> v;
    sort(points.begin(), points.end(), [](vector<int> v1, vector<int> v2)
    {
        return (v1[0]*v1[0] + v1[1]*v1[1]) < (v2[0]*v2[0] + v2[1]*v2[1]);
    });
    for(int i=0; i<k; i++)
        v.push_back(points[i]);
    return v;
}
```

Approach2- We Max Heap. T(n) = O(nlog(k)) S(n) = O(k)

Code-
```
vector<vector<int>> kClosest(vector<vector<int>>& points, int k)
{
    vector<vector<int>> v;
    priority_queue<pair<int, vector<int>>> max_heap;
```



(138)

```
for (int i=0; i<points.size(); i++)
{
    max_heap.push({points[i][0]*points[i][0]+
        points[i][1]*points[i][1], points[i]});
    if(max_heap.size()>k)
        max_heap.pop();
}
while(max_heap.size())
{
    v.push_back(max_heap.top().second);
    max_heap.pop();
}
return v;
}
```

6th Top K Frequent Elements -

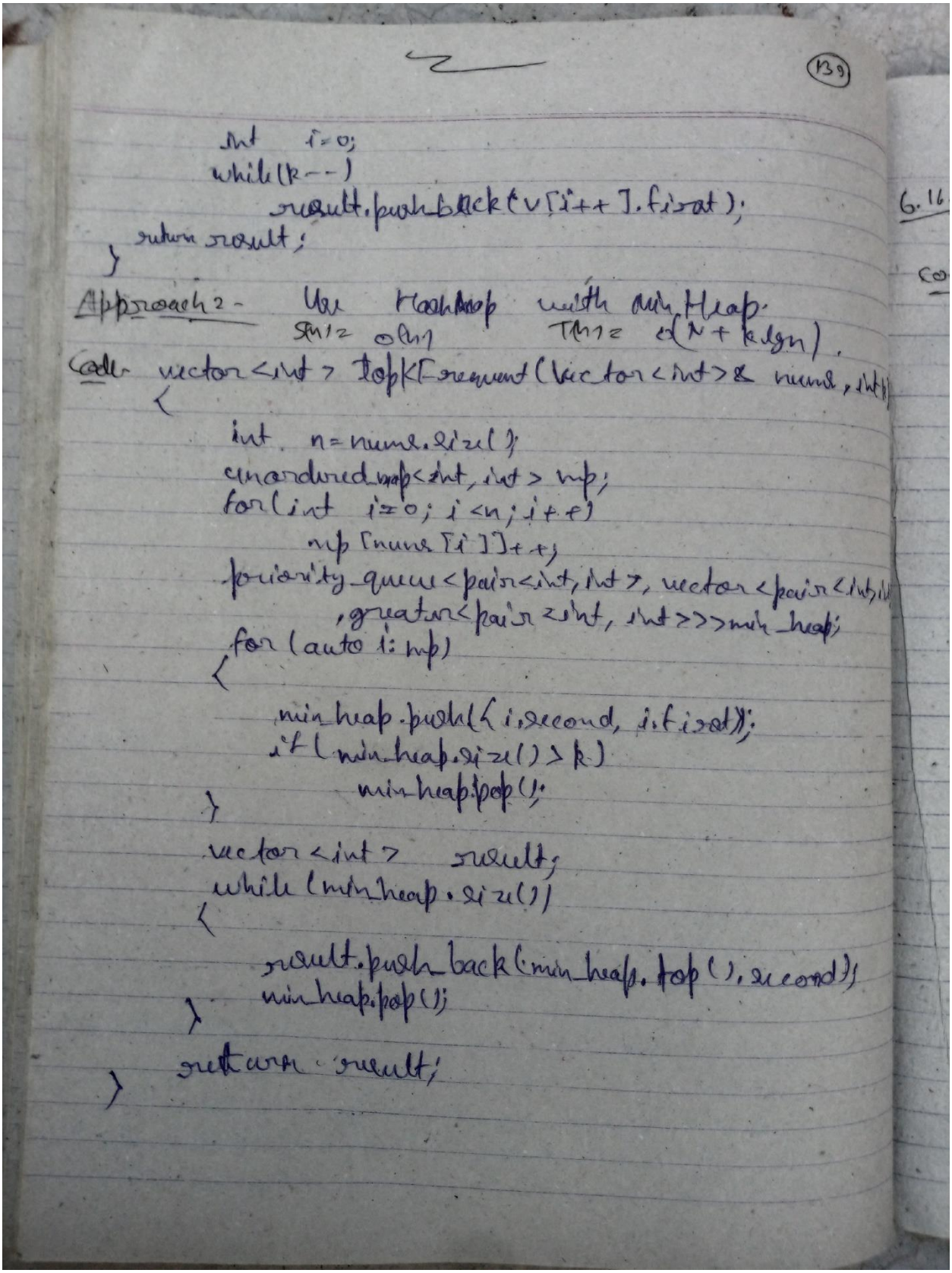
I/P- nums = [1,1,1,2,2,3] k=2.

O/P - [1,2]

Approach :- Use HashMap with Sorting - TM = O(n log n)
SM = O(n)

Code - vector<int> topKFrequent(vector<int>& nums, int k)

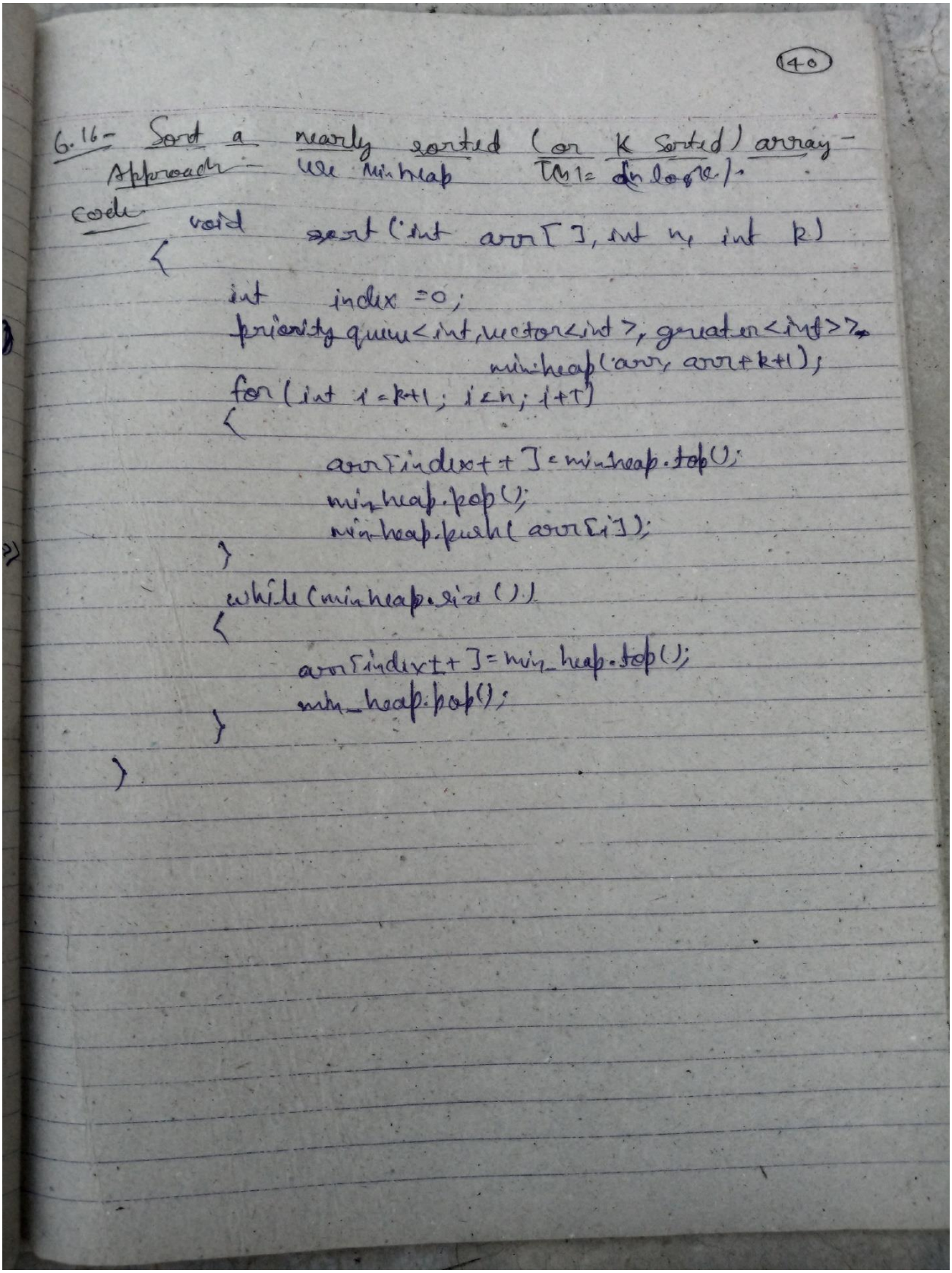
```
{
    int n=nums.size();
    unordered_map<int,int> mp;
    for(int i=0;i<n;i++)
        mp[nums[i]]++;
    vector<pair<int,int>> v(mp.begin(),mp.end());
    vector<int> result;
    sort(v.begin(), v.end(), [](pair<int,int> p1,
        pair<int,int> p2){return p1.second>p2.
        second;});
```



```
    int i=0;
    while(k--)
        result.push_back(v[i++].first);
    return result;
}
```

Approach 2 - Use HashMap with min Heap.
Space O(n) Time O(N + klogn).

```
Code- vector<int> topKFrequent(vector<int>& nums, int k)
{
    int n=nums.size();
    unordered_map<int,int> mp;
    for(int i=0; i<n; i++)
        mp[nums[i]]++;
    priority_queue<pair<int,int>, vector<pair<int,int>>
        ,greater<pair<int,int>>> min_heap;
    for(auto i:mp)
    {
        min_heap.push({i.second, i.first});
        if(min_heap.size()>k)
            min_heap.pop();
    }
    vector<int> result;
    while(min_heap.size())
    {
        result.push_back(min_heap.top().second);
        min_heap.pop();
    }
    return result;
}
```



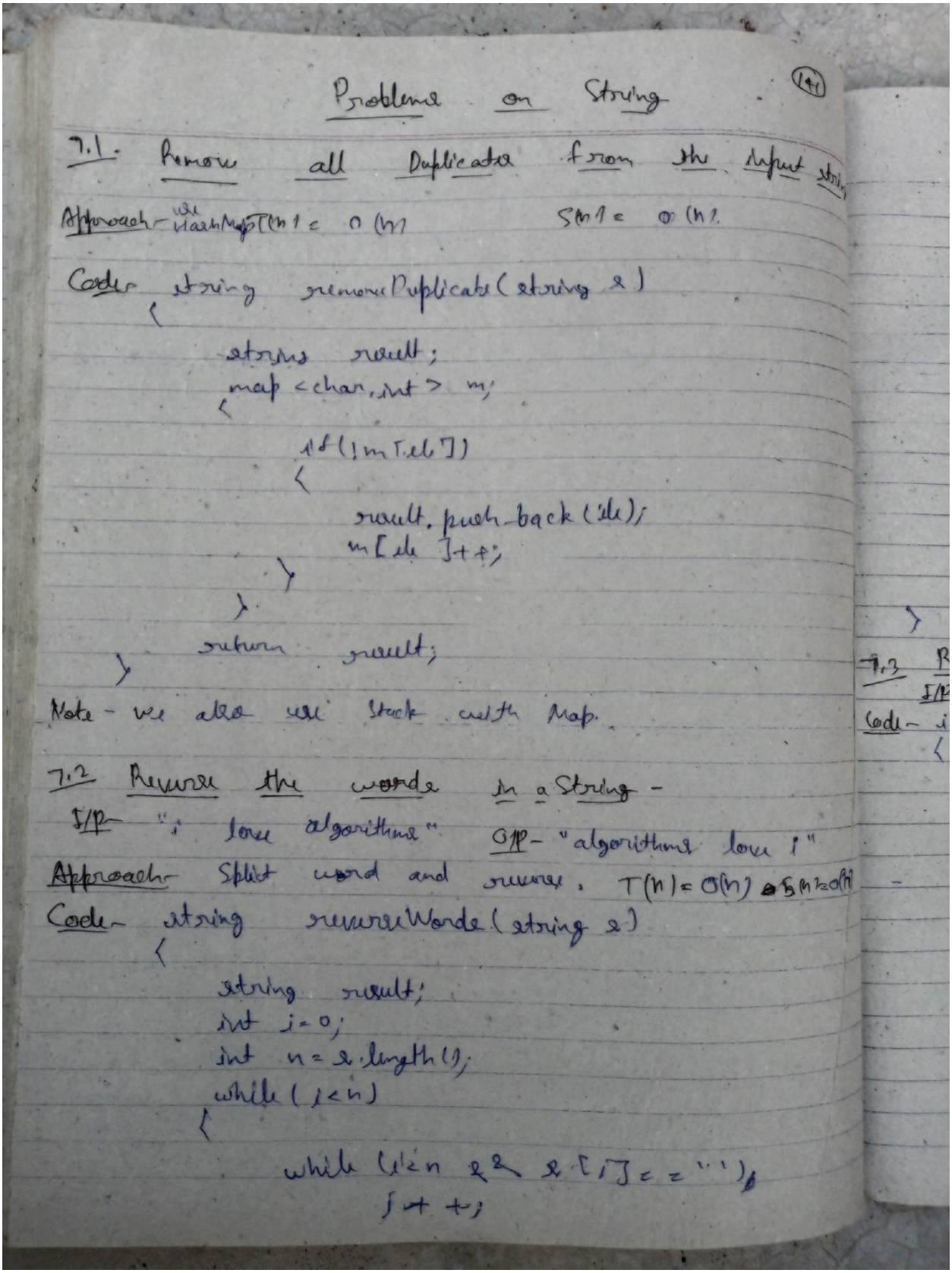
(40)

6.16- Sort a nearly sorted (or K Sorted) array -

Approach - Use min heap T(n) = O(n log k).

Code

```
void sort(int arr[], int n, int k)
{
    int index = 0;
    priority_queue<int, vector<int>, greater<int>>
        min_heap(arr, arr+k+1);
    for(int i=k+1; i<n; i++)
    {
        arr[index++] = min_heap.top();
        min_heap.pop();
        min_heap.push(arr[i]);
    }
    while(min_heap.size())
    {
        arr[index++] = min_heap.top();
        min_heap.pop();
    }
}
```



Problems on String

(141)

7.1 Remove all Duplicates from the input string

Approach - use HashMap T(n) = O(n) S(n) = O(n)

Code -

```
string removeDuplicate(string s)
{
    string result;
    map<char, int> m;
    {
        if(!m[s.el])
        {
            result.push_back(s[i]);
            m[s[i]]++;
        }
    }
    return result;
}
```

Note - we also use Stack with Map.

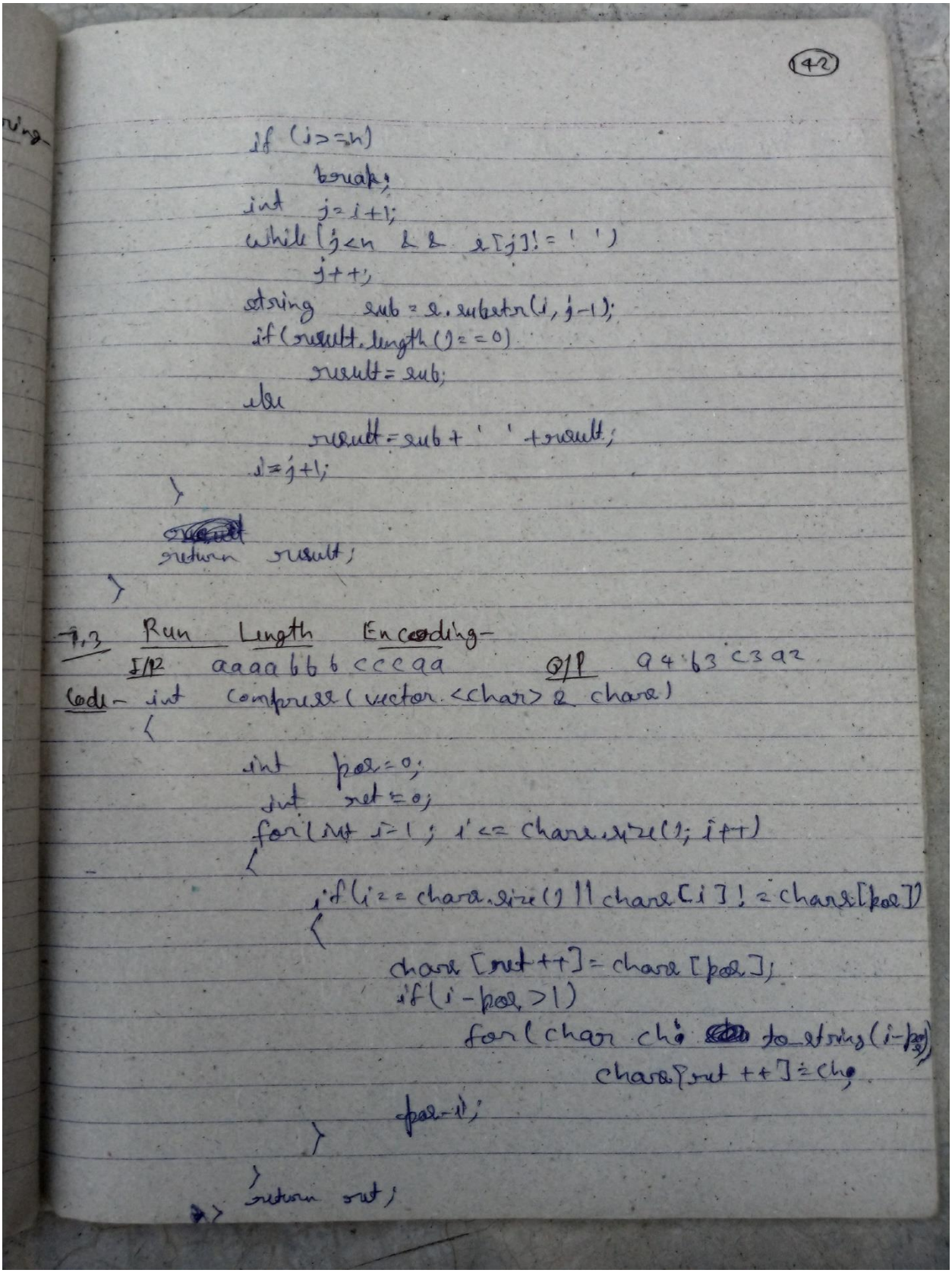
7.2 Reverse the words in a String -

I/P - "i love algorithms" O/P - "algorithms love i"

Approach - Split word and reverse. T(n) = O(n) S(n) = O(n)

Code -

```
string reverseWords(string s)
{
    string result;
    int i = 0;
    int n = s.length();
    while(i<n)
    {
        while(i<n && s[i]==' ')
            i++;
```



(42)

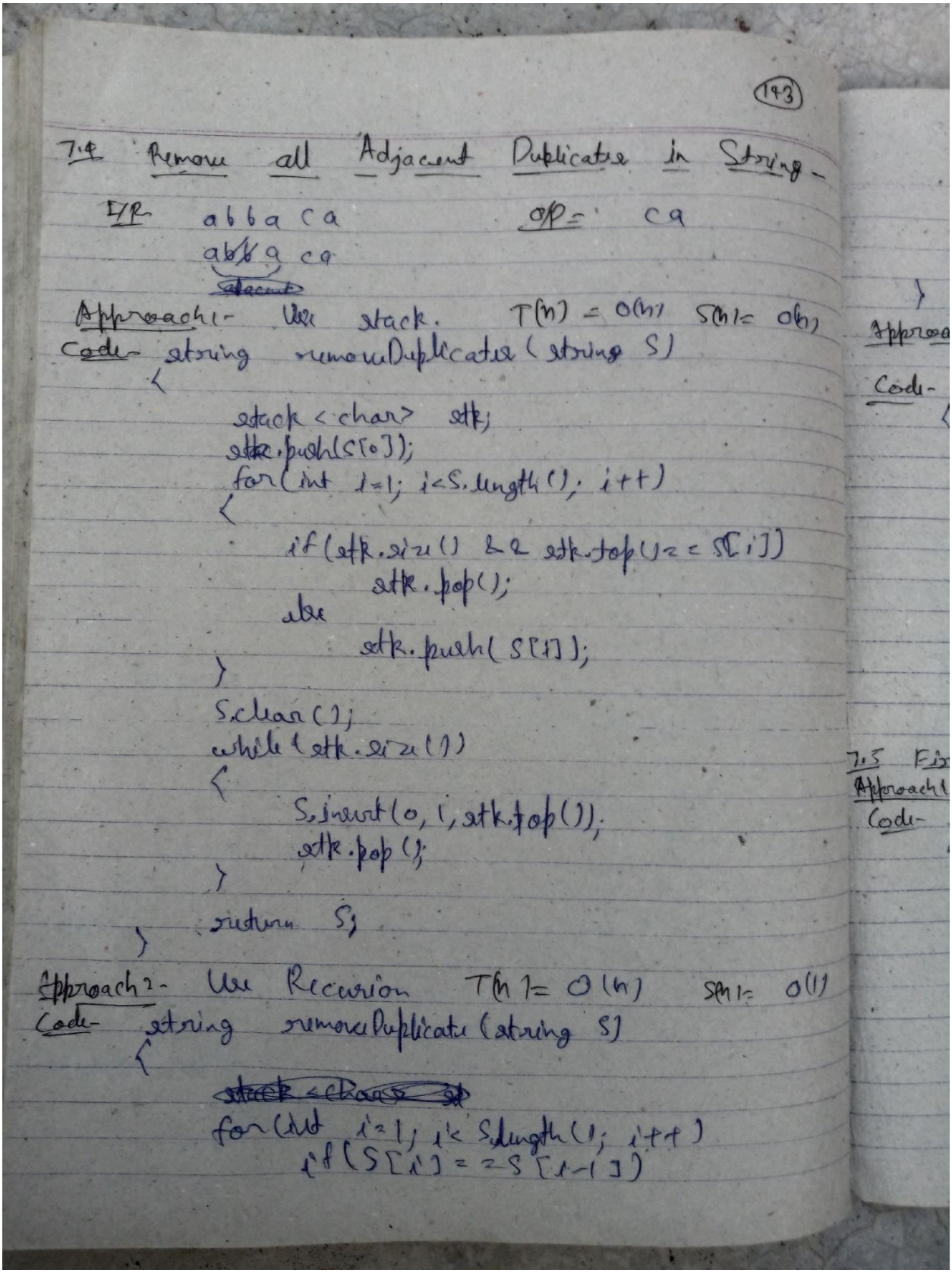
```
if (i>=n)
    break;
int j=i+1;
while(j<n && s[j]!=' ')
    j++;
string sub = s.substr(i, j-1);
if(result.length()==0)
    result=sub;
else
    result=sub+' '+result;
i=j+1;
}
return result;
}
```

7.3 Run Length Encoding-

I/P aaaabbbccccaa O/P a4b3c3a2

Code-

```
int compress(vector<char>& chars)
{
    int pos=0;
    int ret=0;
    for(int i=1; i<=chars.size(); i++)
    {
        if(i==chars.size() || chars[i]!=chars[pos])
        {
            chars[ret++]=chars[pos];
            if(i-pos>1)
                for(char ch : to_string(i-pos))
                    chars[ret++]=ch;
            pos=i;
        }
    }
    return ret;
}
```



(143)

7.4 Remove all Adjacent Duplicates in String -

I/P abbaca o/p = ca

abbaca

adjacent

Approach1- Use stack. T(n) = O(n) S(n) = O(n)

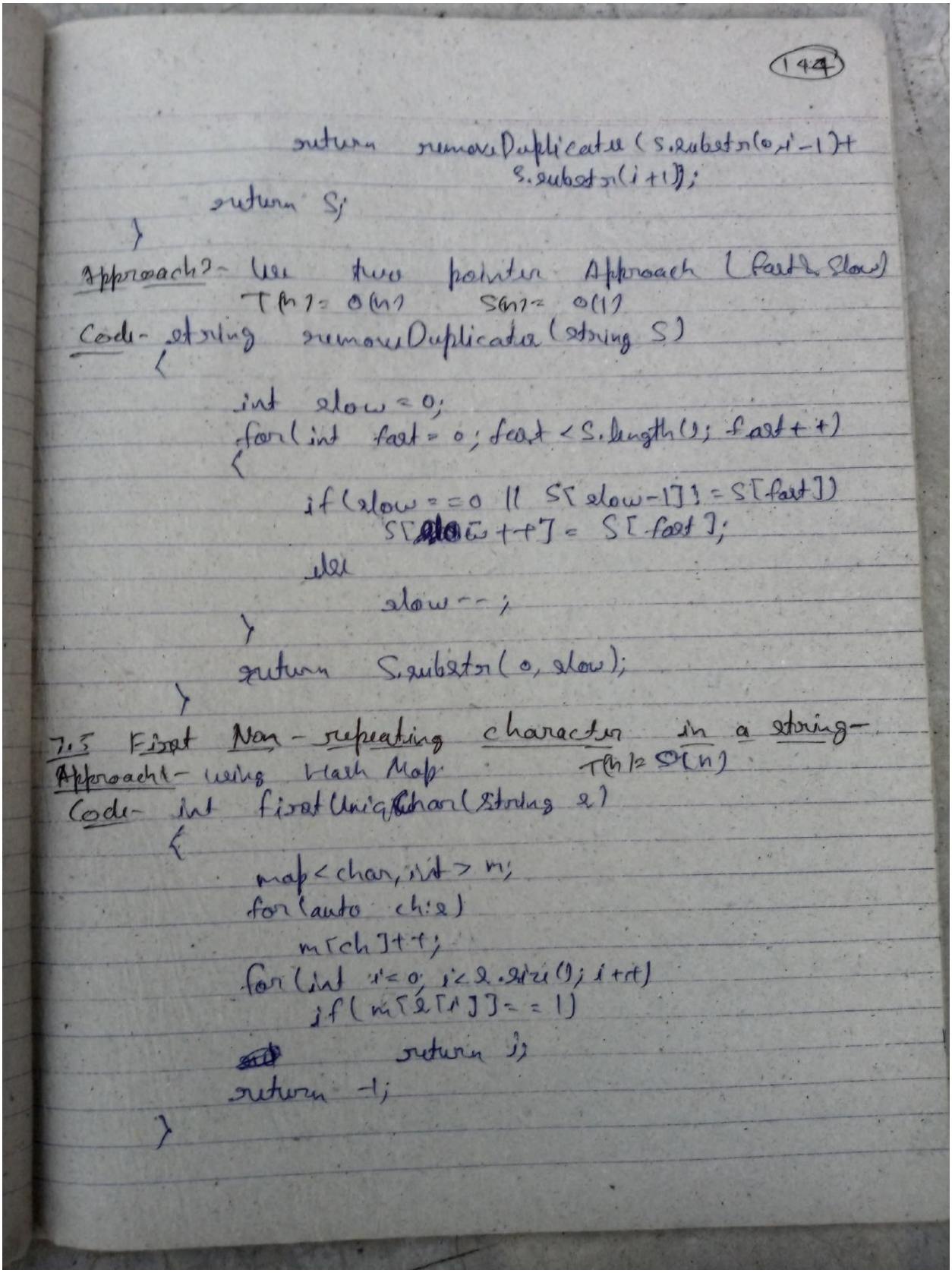
Code-

```
string removeDuplicates(string S)
{
    stack<char> stk;
    stk.push(S[0]);
    for(int i=1; i<S.length(); i++)
    {
        if(stk.size() && stk.top()==S[i])
            stk.pop();
        else
            stk.push(S[i]);
    }
    S.clear();
    while(stk.size())
    {
        S.insert(0, 1, stk.top());
        stk.pop();
    }
    return S;
}
```

Approach2- Use Recursion T(n) = O(n) S(n) = O(1)

Code-

```
string removeDuplicate(string S)
{
    stack<char> st
    for(int i=1; i<S.length(); i++)
        if(S[i]==S[i-1])
```



(144)

```
        return removeDuplicates(s.substr(0,i-1)+
                                s.substr(i+1));
    return s;
}
```

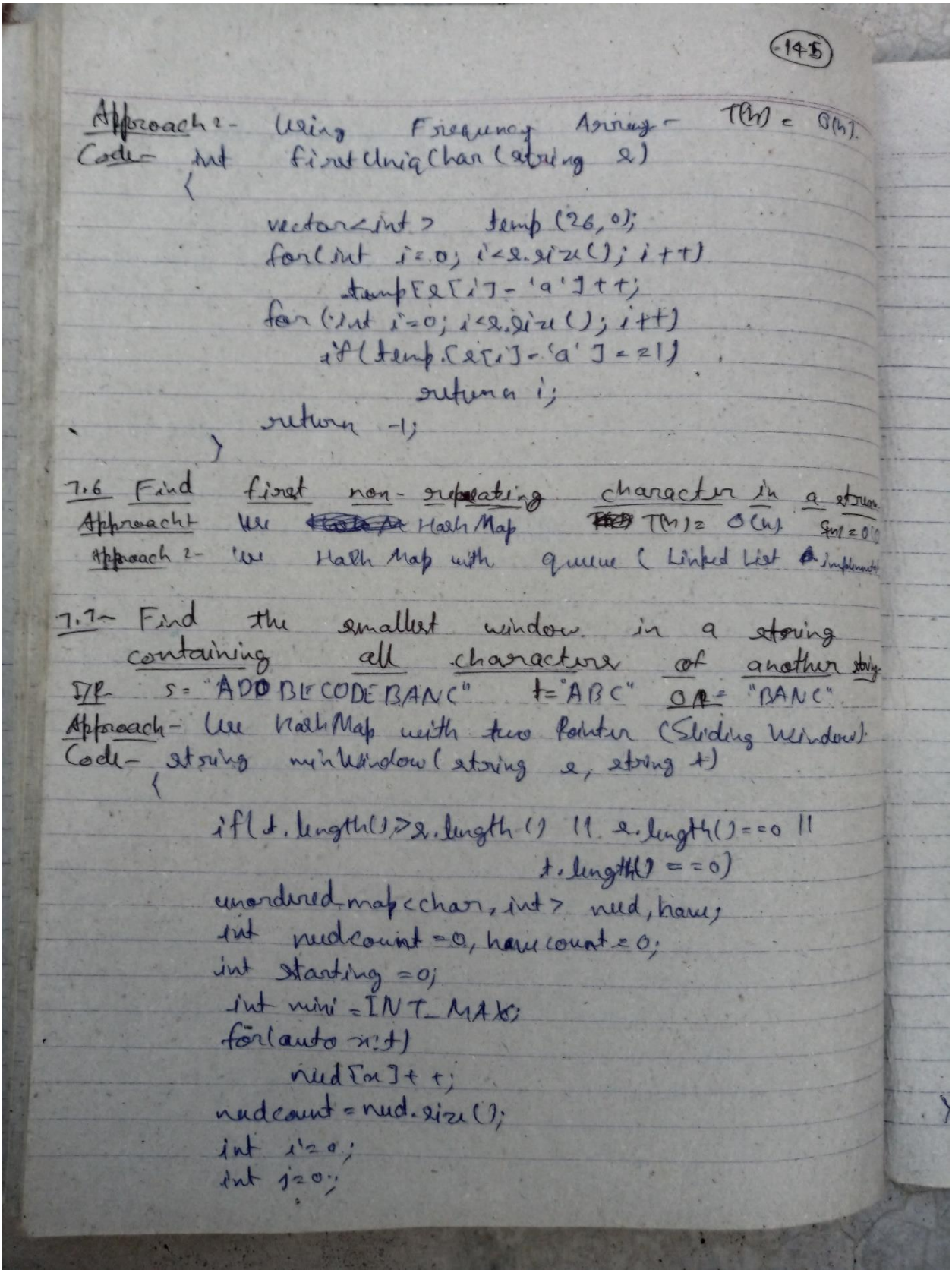
Approach2 - Use two pointer Approach (fast & slow)
T(n) = O(n) S(n) = O(1)

Code - string removeDuplicates(string s)

```
{
    int slow = 0;
    for(int fast = 0; fast < s.length(); fast++)
    {
        if(slow == 0 || s[slow-1] != s[fast])
            s[slow++] = s[fast];
        else
            slow--;
    }
    return s.substr(0, slow);
}
```

7.5 First Non-repeating character in a string -
Approach1 - using Hash Map T(n) = O(n)
Code - int firstUniqChar(string s)

```
{
    map<char, int> m;
    for(auto ch:s)
        m[ch]++;
    for(int i=0; i<s.size(); i++)
        if(m[s[i]] == 1)
            return i;
    return -1;
}
```



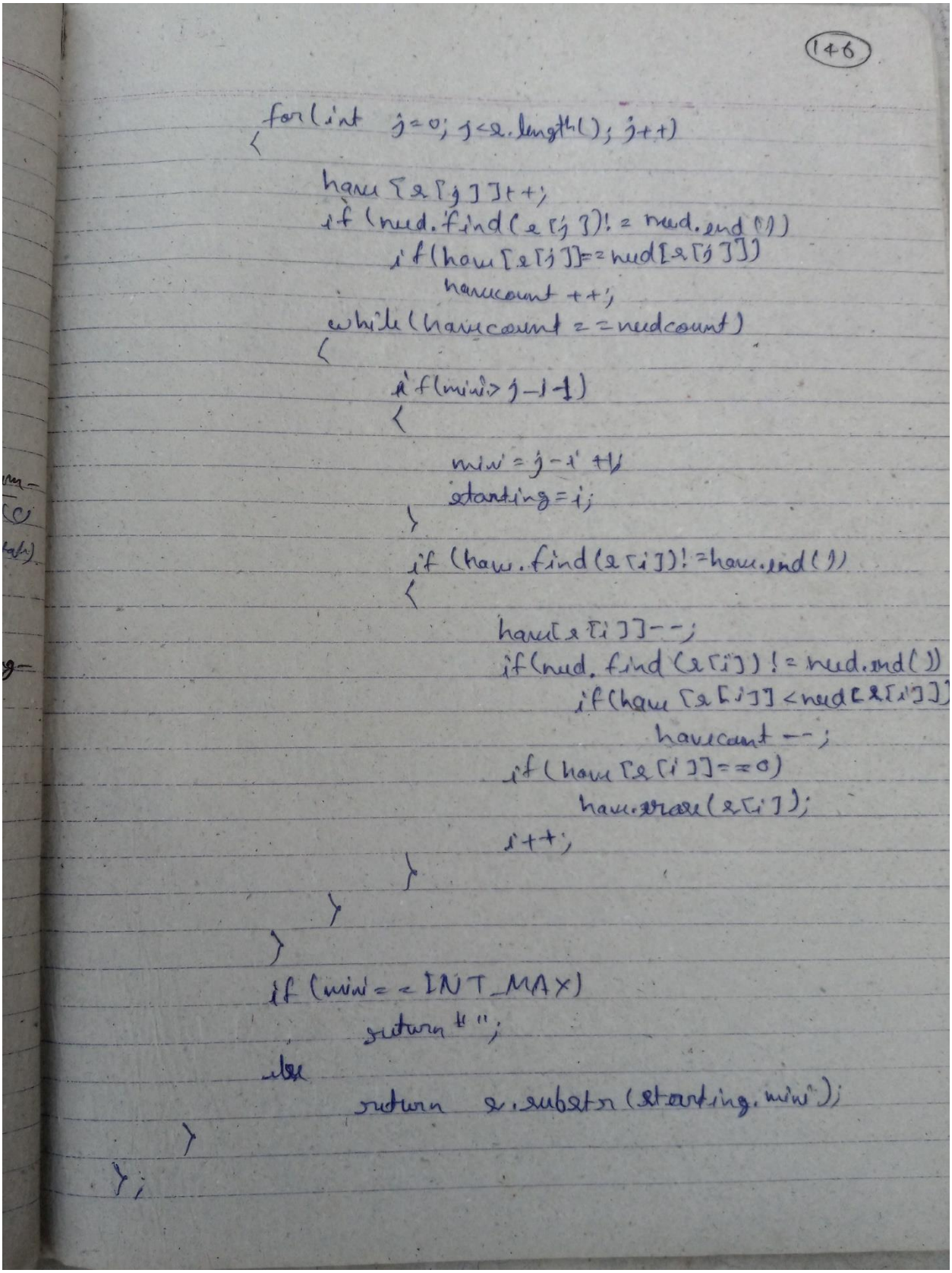
(145)

Approach:- Using Frequency Array - T(N) = O(N).

Code- int firstUniqChar(string s)
{
vector<int> temp(26,0);
for(int i=0; i<s.size(); i++)
temp[s[i]-'a']++;
for(int i=0; i<s.size(); i++)
if(temp[s[i]-'a']==1)
return i;
return -1;
}

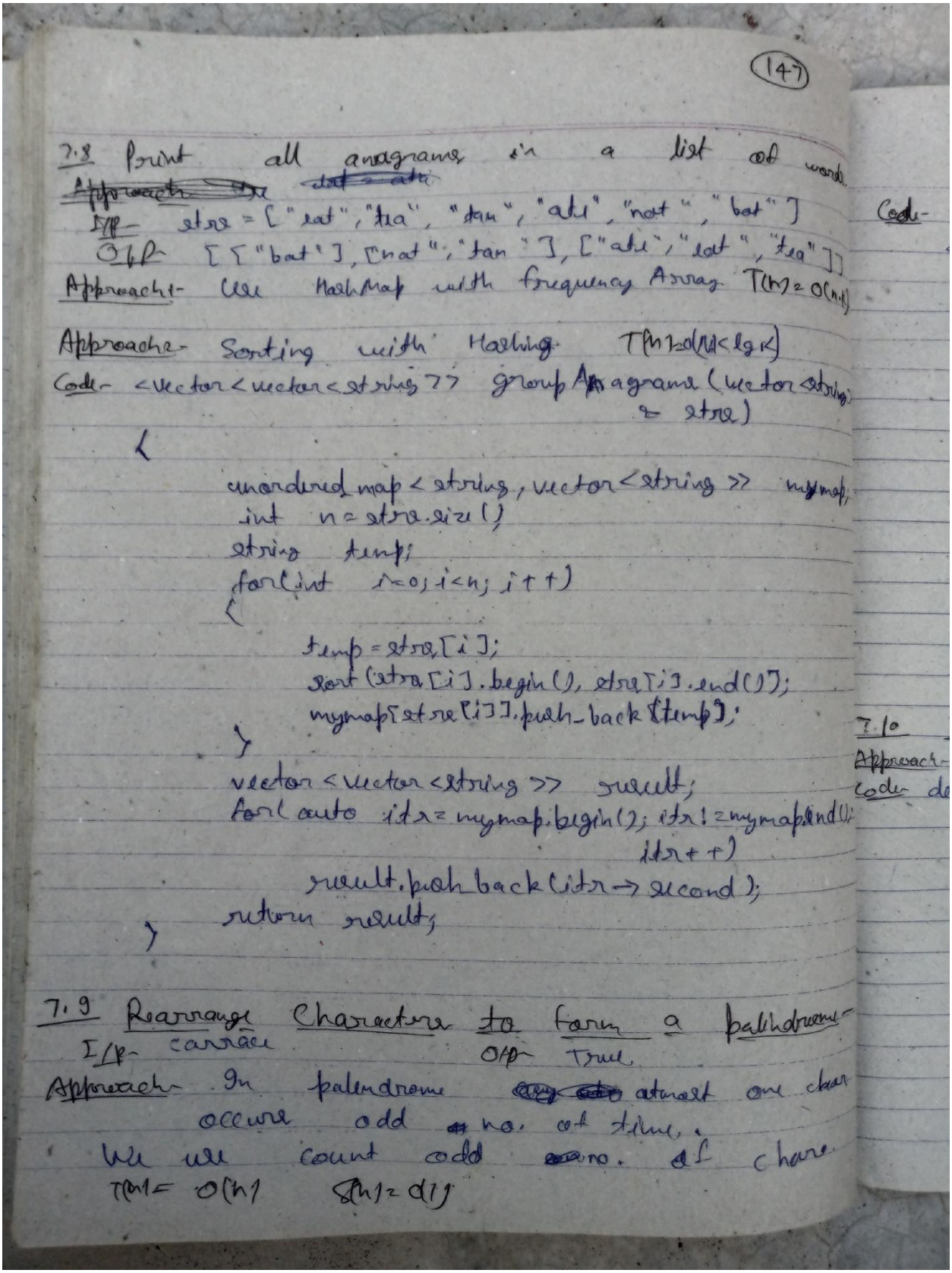
7.6 Find first non-repeating character in a stream.
Approach1- We ~~Code~~ Hash Map ~~FIFO~~ T(N) = O(N) Sp = O(1)
Approach 2- We Hash Map with queue (Linked List ~~A~~ Implementation)

7.7- Find the smallest window in a string containing all characters of another string.
I/P- S = "ADOBECODEBANC" t = "ABC" O/P = "BANC"
Approach- Use HashMap with two Pointer (Sliding Window).
Code- string minWindow(string s, string t)
{
if(t.length()>s.length() || s.length()==0 ||
t.length()==0)
unordered_map<char,int> need,have;
int needCount=0, haveCount=0;
int starting=0;
int mini=INT_MAX;
for(auto x:t)
need[x]++;
needCount=need.size();
int i=0;
int j=0;



```
(146)

for(int j=0; j<s.length(); j++)
{
    have[s[j]]++;
    if(need.find(s[j])!=need.end())
        if(have[s[j]]==need[s[j]])
            havecount++;
    while(havecount==needcount)
    {
        if(min>j-i+1)
        {
            min=j-i+1;
            starting=i;
        }
        if(have.find(s[i])!=have.end())
        {
            have[s[i]]--;
            if(need.find(s[i])!=need.end())
                if(have[s[i]]<need[s[i]])
                    havecount--;
            if(have[s[i]]==0)
                have.erase(s[i]);
            i++;
        }
    }
}
if(min==INT_MAX)
    return "";
else
    return s.substr(starting,min);
}
};
```



(147)

7.8 Print all anagrams in a list of words.

I/P- str = ["eat", "tea", "tan", "ate", "nat", "bat"]
O/P- [["bat"], ["nat", "tan"], ["ate", "eat", "tea"]]

Approach:- Use HashMap with frequency Array T(n) = O(n.k)

Approach:- Sorting with Hashing T(n) = O(nklogk)

Code-

```
vector<vector<string>> groupAnagrams(vector<string>& strs)
{
    unordered_map<string, vector<string>> mymap;
    int n = strs.size();
    string temp;
    for(int i=0; i<n; i++)
    {
        temp = strs[i];
        sort(strs[i].begin(), strs[i].end());
        mymap[strs[i]].push_back(temp);
    }
    vector<vector<string>> result;
    for(auto itr = mymap.begin(); itr != mymap.end();
                                        itr++)
        result.push_back(itr->second);
    return result;
}
```

7.9 Rearrange Characters to form a palindrome-

I/P- carrace O/P- True

Approach- In palindrome atmost one char occurs odd no. of time.

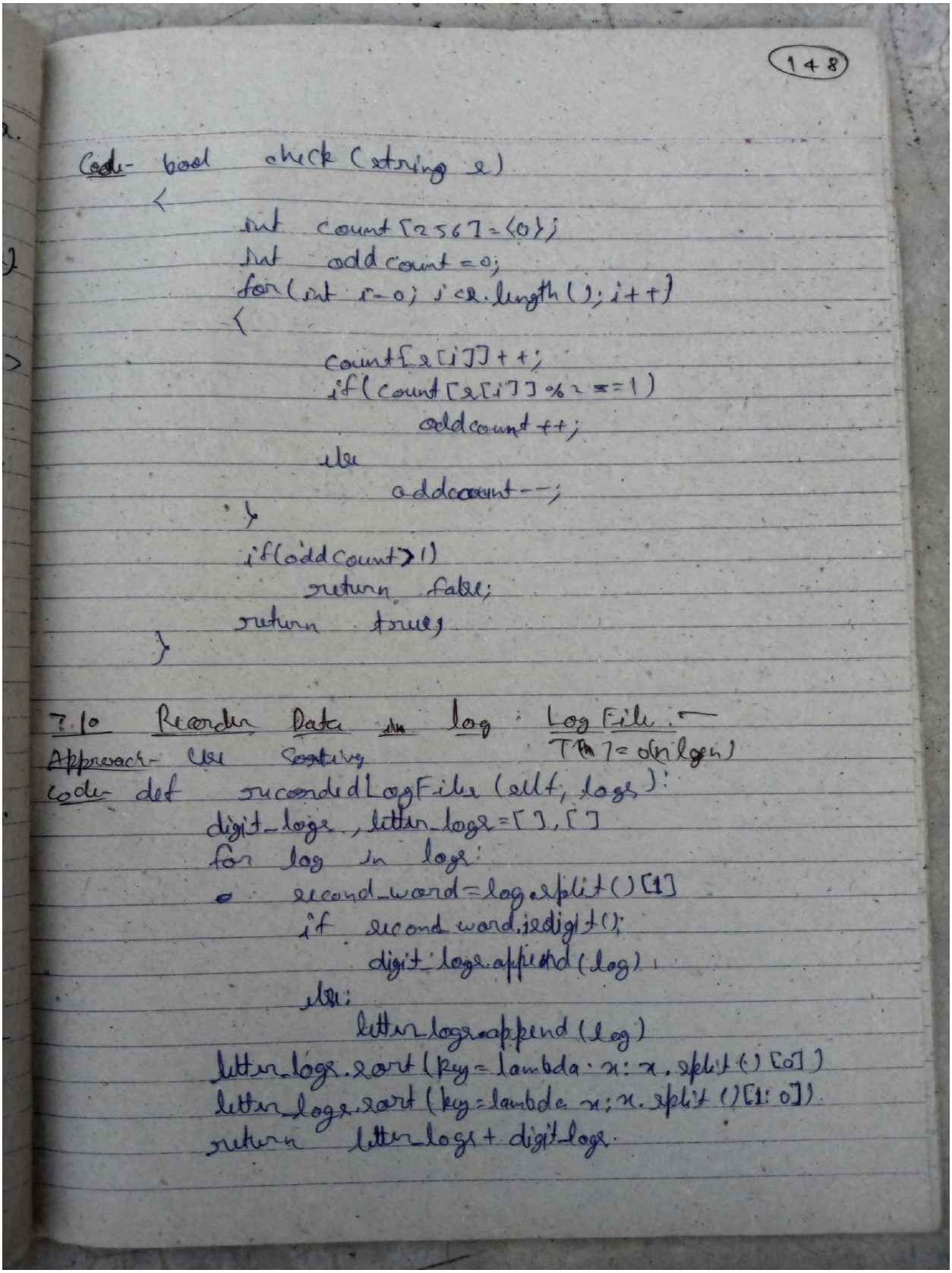
We will count odd no. of chars.

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

7.10

Approach-

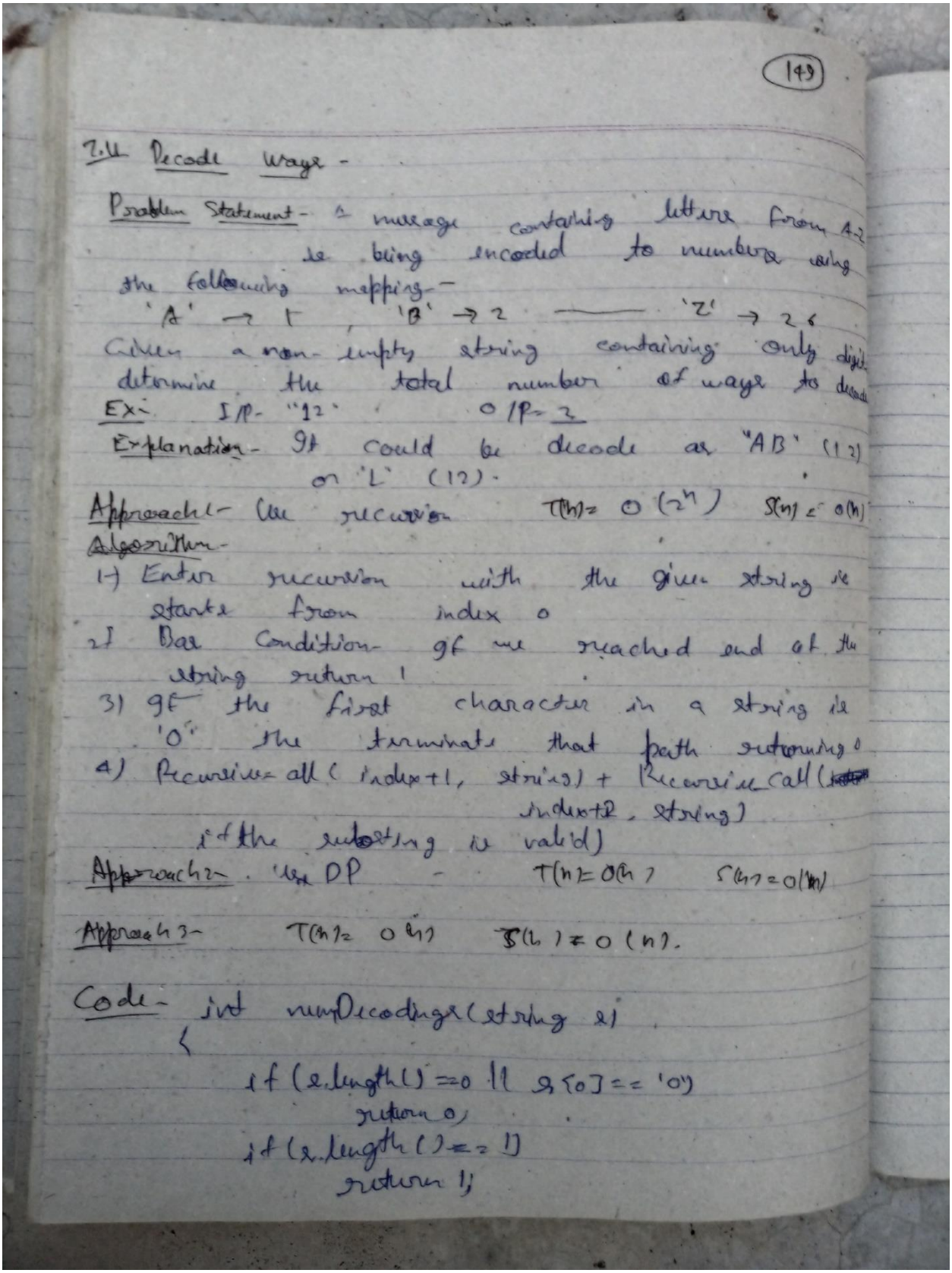
Code- do



148

Code- bool check (string s)
{
int count[256]={0};
int oddcount=0;
for(int i=0; i<s.length(); i++)
{
countf[s[i]]++;
if(count[s[i]]%2==1)
oddcount++;
else
oddcount--;
}
if(oddcount>1)
return false;
return true;
}

7.10 Reorder Data in log : Log File :-
Approach- Use Sorting T(n) = O(n log n)
Code- def reconderdLogFiles(self, logs):
digit_logs, letter_logs=[],[]
for log in logs:
second_word=log.split()[1]
if second word.isdigit():
digit_logs.append(log)
else:
letter_logs.append(log)
letter_logs.sort(key=lambda x: x.split()[0])
letter_logs.sort(key=lambda x: x.split()[1:0])
return letter_logs+digit_logs.



(149)

7.11 Decode Ways -

Problem Statement - A message containing letters from A-Z is being encoded to numbers using the following mapping -

'A' → 1, 'B' → 2 ——— 'Z' → 26

Given a non-empty string containing only digits determine the total number of ways to decode

Ex- I/P- "12" O/P- 2

Explanation - It could be decode as "AB" (1 2) or "L" (12).

Approach1 - Use recursion T(n) = O(2^n) S(n) = O(n)

Algorithm -

1) Enter recursion with the given string we starts from index 0

2) Base Condition - If we reached end of the string return 1

3) If the first character in a string is '0' the terminate that path returning 0

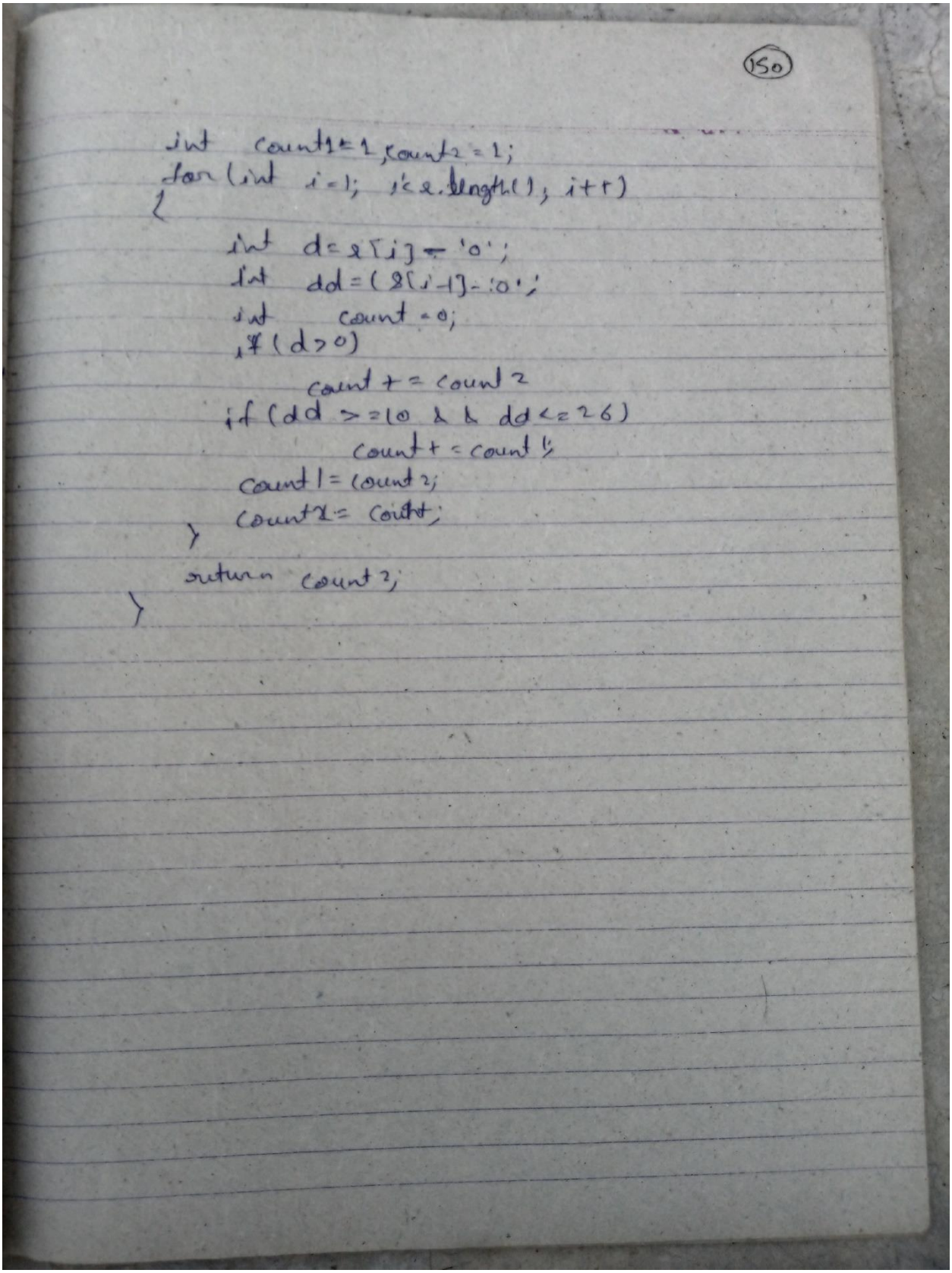
4) Recursive call (index+1, string) + Recursive call (index+2, string) if the substring is valid)

Approach2 - Use DP - T(n) = O(n) S(n) = O(n)

Approach3 - T(n) = O(n) S(n) = O(n).

Code -

```
int numDecodings(string s)
{
    if (s.length() == 0 || s[0] == '0')
        return 0;
    if (s.length() == 1)
        return 1;
```



(150)

```
int count1=1,count2=1;
for(int i=1; i<s.length(); i++)
{
    int d=s[i]-'0';
    int dd=(s[i-1]-'0';
    int count=0;
    if(d>0)
        count+=count2
    if(dd>=10 && dd<=26)
        count+=count1;
    count1=count2;
    count2=count;
}
return count2;
}
```

(151)

7.15 Longest Common Prefix -

Ip- strs = ["flower", "flow", "flight"] Op - "fl"

Approach:- Use ~~two~~ nested loop T(n) = O(MXL) Space O(1)

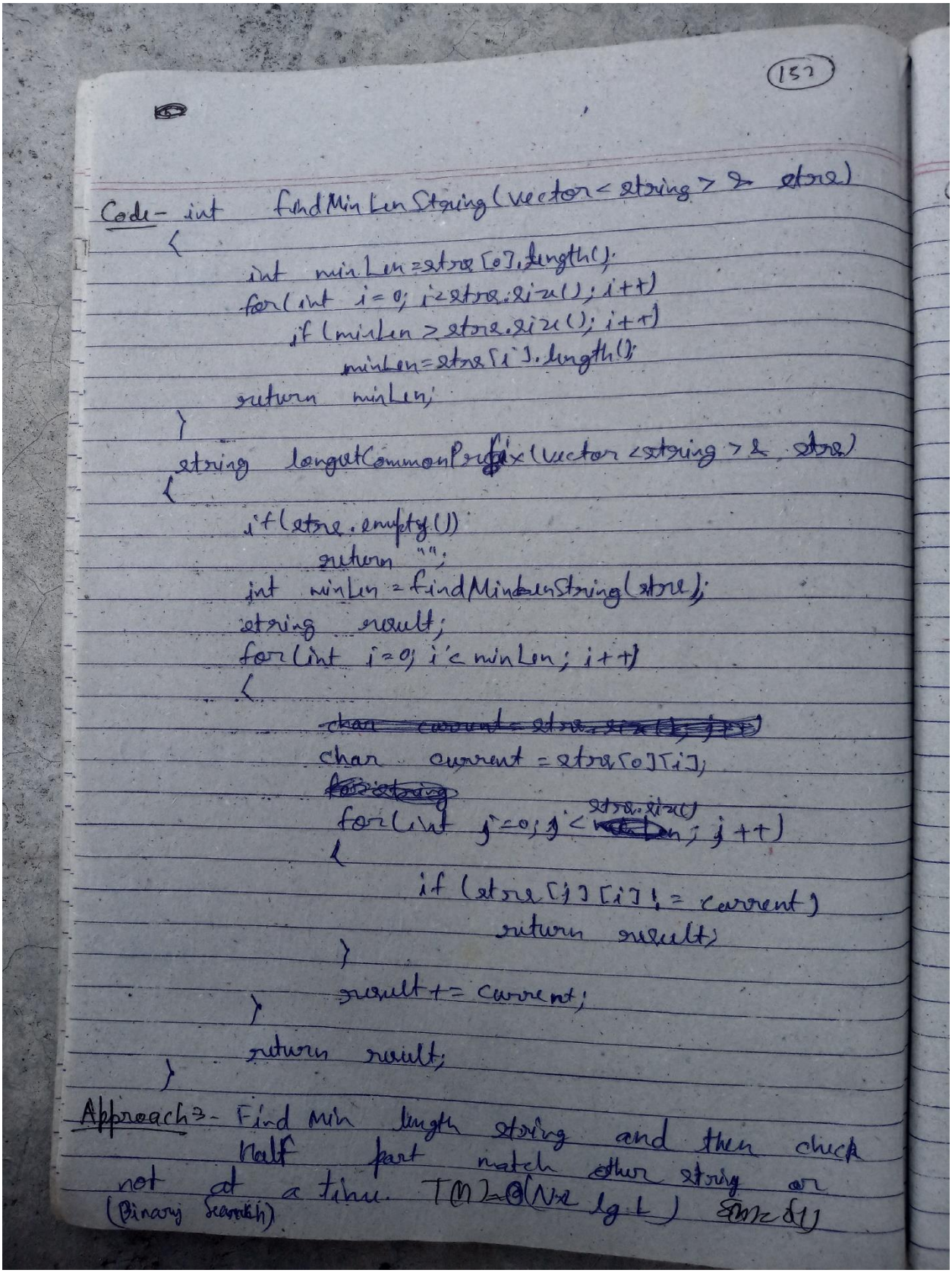
Code- string commonPrefixUtil(string str1, string str2)

```
{
    int n1 = str1.length();
    int n2 = str2.length();
    int i=0, j=0;
    while (i<n1 && j<n2)
    {
        if(str1[i] != str2[j])
            break;
        i++;
        j++;
    }
    return str1.substr(0,i);
}
string longestCommonPrefix(vector<string>& strs)
{
    if(strs.empty())
        return "";
    string prefix = strs[0];
    for(int i=1; i<strs.size(); i++)
        prefix = commonPrefixUtil(prefix, strs[i]);
    return prefix;
}
```

Check word to word

Approach:- Check char. to char. O(n)

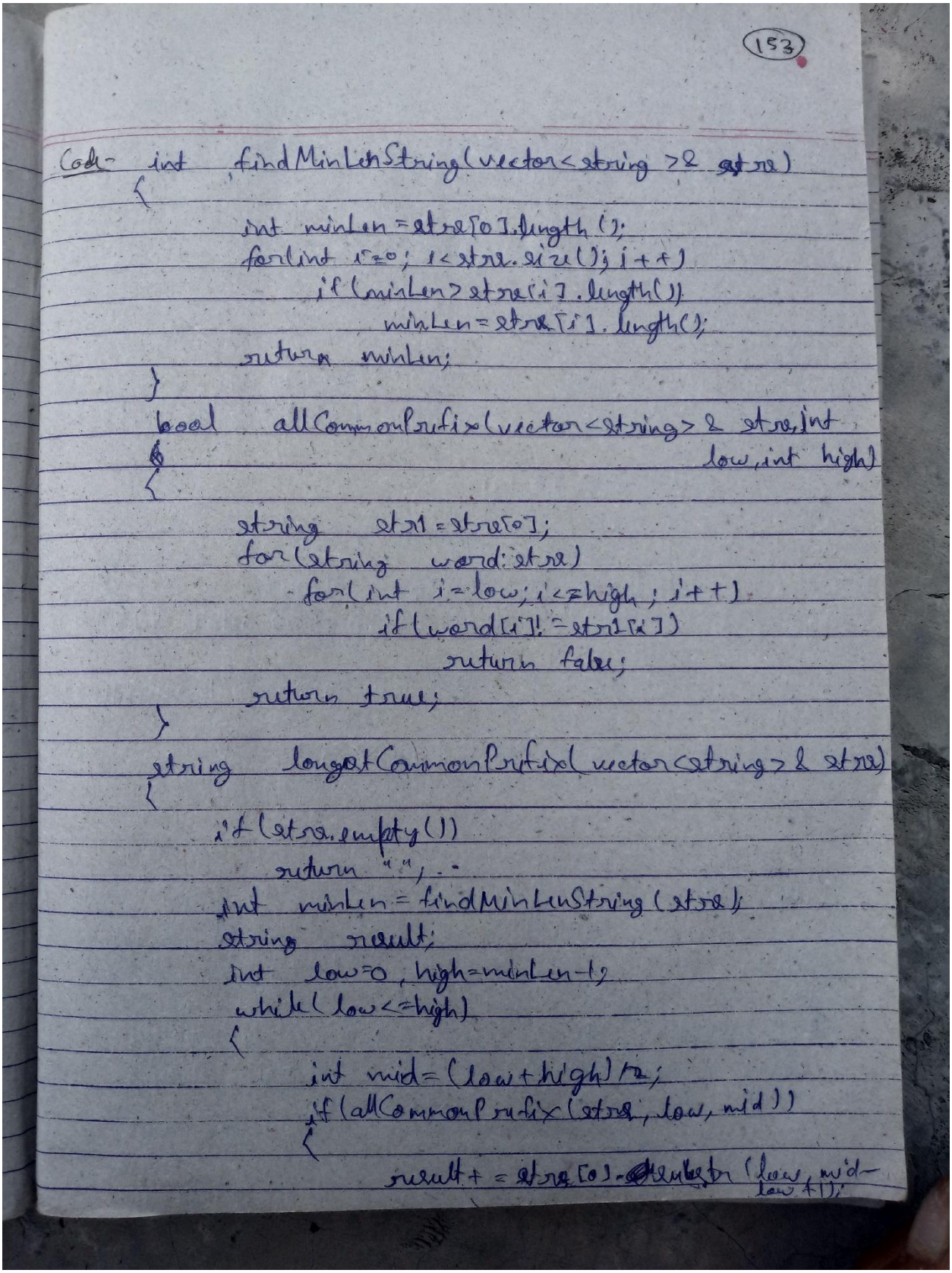
T(n) < O(NXL) S(n) = O(1)



(157)

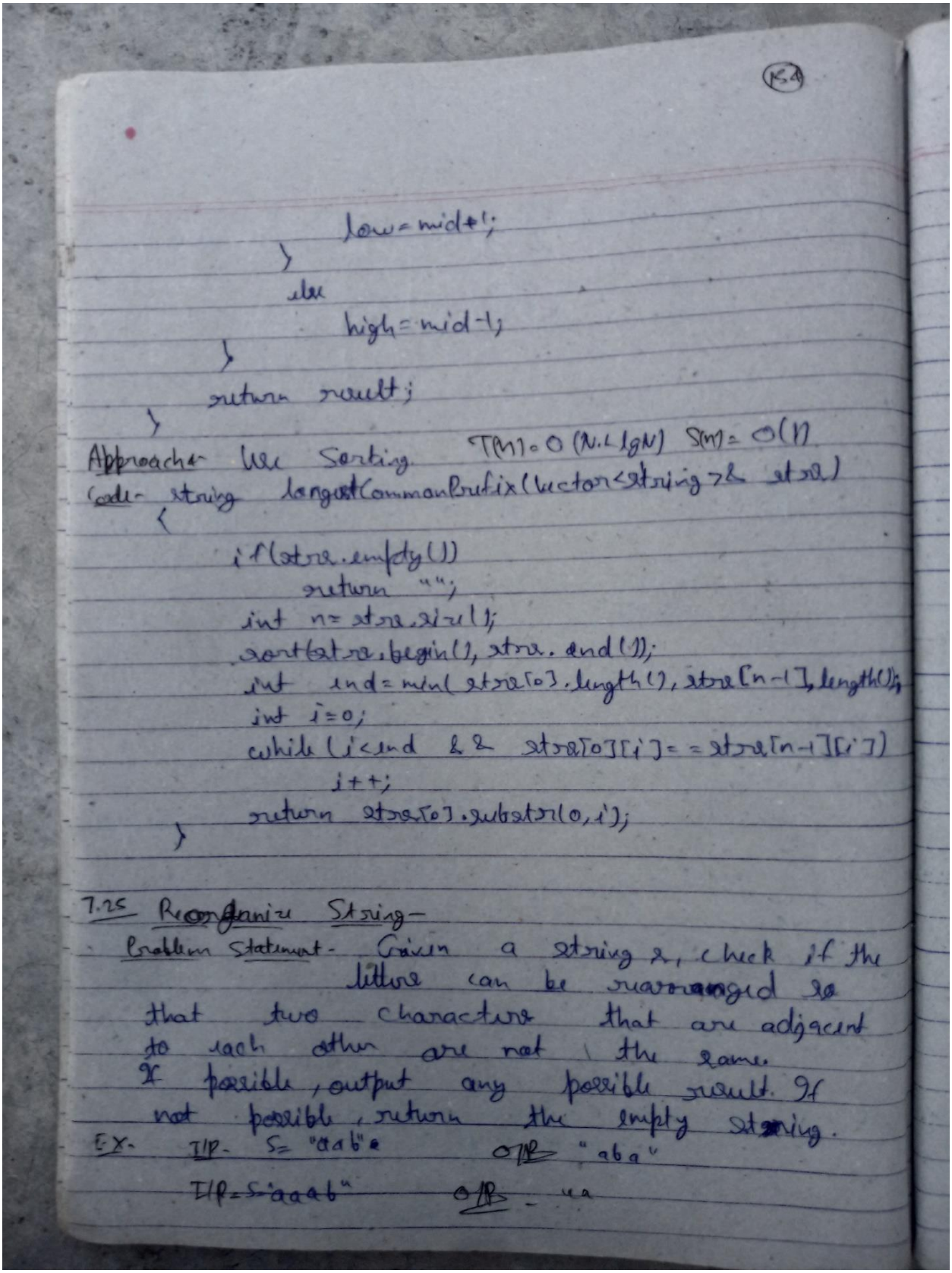
Code- int findMinLenString(vector<string>& strs)
{
int minLen = strs[0].length();
for(int i=0; i<strs.size(); i++)
if(minLen > strs[i].size(); i++)
minLen = strs[i].length();
return minLen;
}
string longestCommonPrefix(vector<string>& strs)
{
if(strs.empty())
return "";
int minLen = findMinLenString(strs);
string result;
for(int i=0; i<minLen; i++)
{
char current = strs[0][i];
for(int j=0; j<strs.size(); j++)
{
if(strs[j][i] != current)
return result;
}
result += current;
}
return result;
}

Approach:- Find min length string and then check half part match other string or not at a time. TC = O(N*lg L) SC = O(1) (Binary search)



153

```
Code- int findMinLenString(vector<string>& strs)
{
    int minLen = strs[0].length();
    for(int i=0; i<strs.size(); i++)
        if(minLen > strs[i].length())
            minLen = strs[i].length();
    return minLen;
}
bool allCommonPrefix(vector<string>& strs, int
                                    low, int high)
{
    string str1 = strs[0];
    for(string word: strs)
        for(int i=low; i<=high; i++)
            if(word[i] != str1[i])
                return false;
    return true;
}
string longestCommonPrefix(vector<string>& strs)
{
    if(strs.empty())
        return "";
    int minLen = findMinLenString(strs);
    string result;
    int low=0, high=minLen-1;
    while(low<=high)
    {
        int mid=(low+high)/2;
        if(allCommonPrefix(strs, low, mid))
        {
            result += strs[0].substr(low, mid-
                                    low+1);
```



```
            low = mid+1;
        }
        else
            high = mid-1;
    }
    return result;
}
```

Approach:- Use Sorting. T[n] = O(N.L lgN) S[n] = O(1)

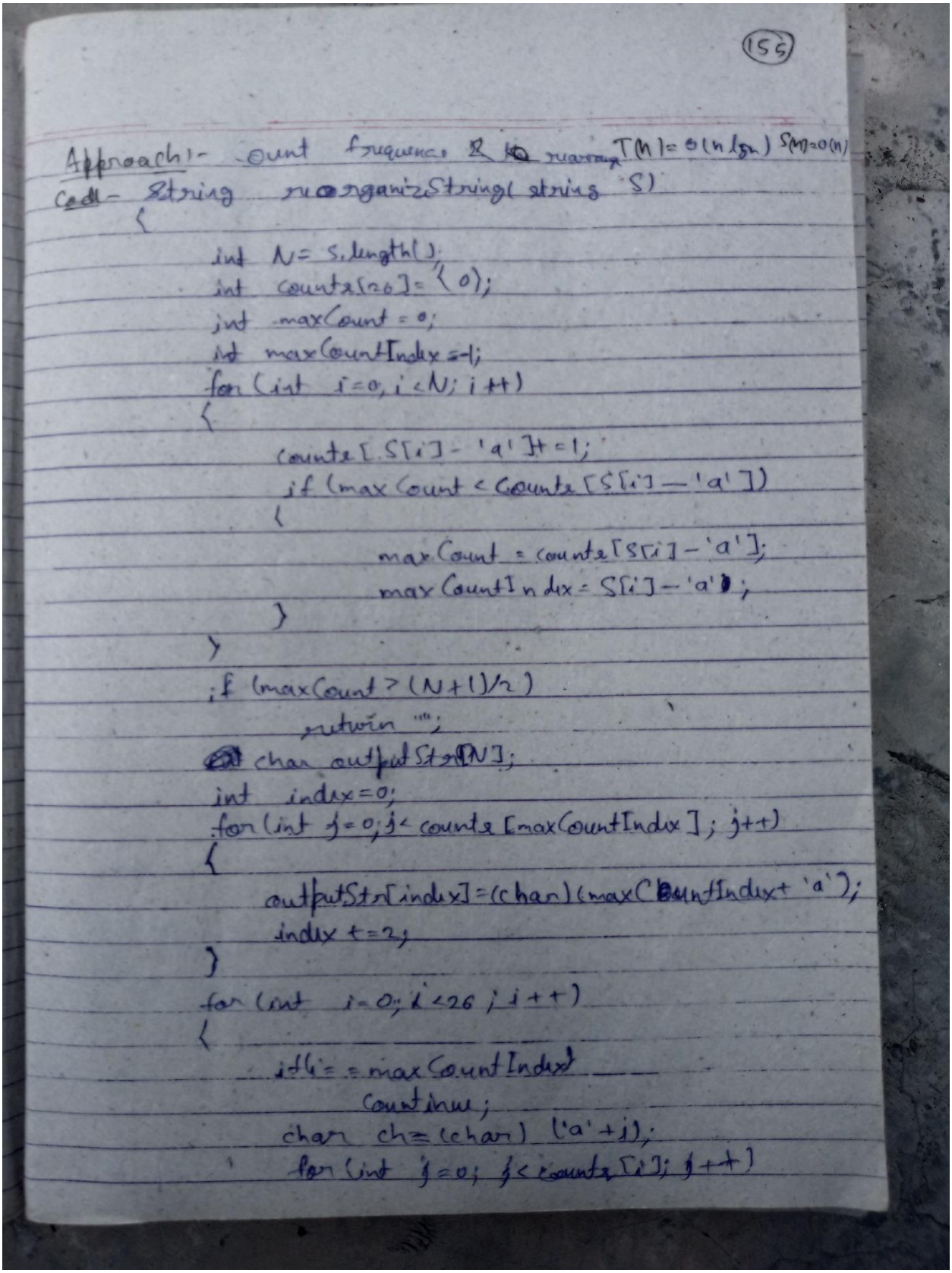
Code:-
```
string longestCommonPrefix(vector<string>& strs)
{
    if(strs.empty())
        return "";
    int n = strs.size();
    sort(strs.begin(), strs.end());
    int end = min(strs[0].length(), strs[n-1].length());
    int i=0;
    while(i<end && strs[0][i] == strs[n-1][i])
        i++;
    return strs[0].substr(0,i);
}
```

7.25 Reorganize String-

Problem Statement- Given a string s, check if the letters can be rearranged so that two characters that are adjacent to each other are not the same.
If possible, output any possible result. If not possible, return the empty string.

Ex. I/P- S= "aab" O/P "aba"

I/P = S= "aaab" O/P - ""

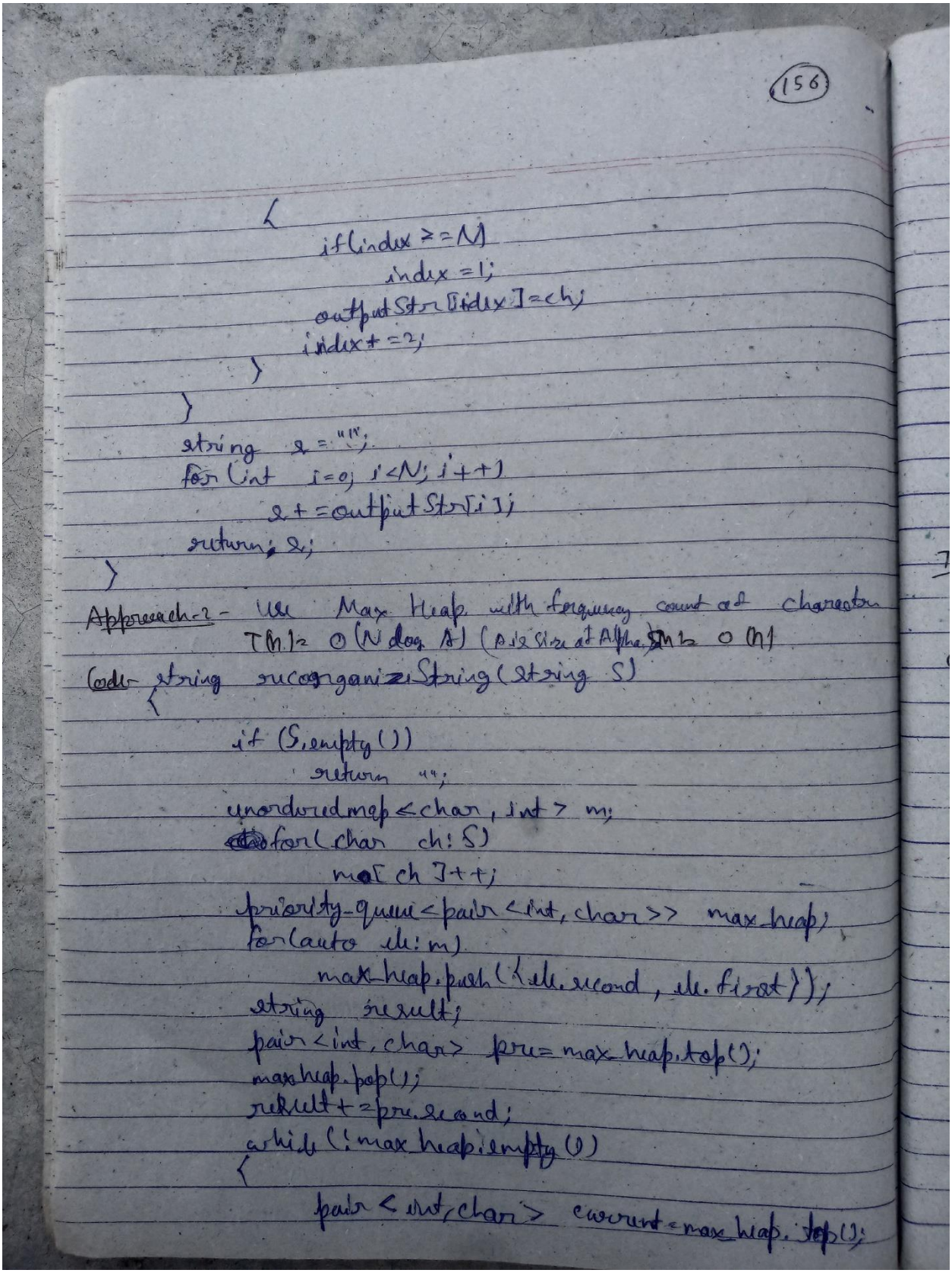


155

Approach:- Count frequency & T(n) = O(n log n) S(n) = O(n)

Code- String reorganizeString(string S)

```
{
    int N = S.length();
    int counts[26] = {0};
    int maxCount = 0;
    int maxCountIndex = -1;
    for (int i=0; i<N; i++)
    {
        counts[S[i] - 'a']++;
        if (maxCount < counts[S[i] - 'a'])
        {
            maxCount = counts[S[i] - 'a'];
            maxCountIndex = S[i] - 'a';
        }
    }
    if (maxCount > (N+1)/2)
        return "";
    char outputStr[N];
    int index=0;
    for (int j=0; j< counts[maxCountIndex]; j++)
    {
        outputStr[index] = (char)(maxCountIndex + 'a');
        index += 2;
    }
    for (int i=0; i<26; i++)
    {
        if(i == maxCountIndex)
            continue;
        char ch = (char)('a'+i);
        for (int j=0; j< counts[i]; j++)
```

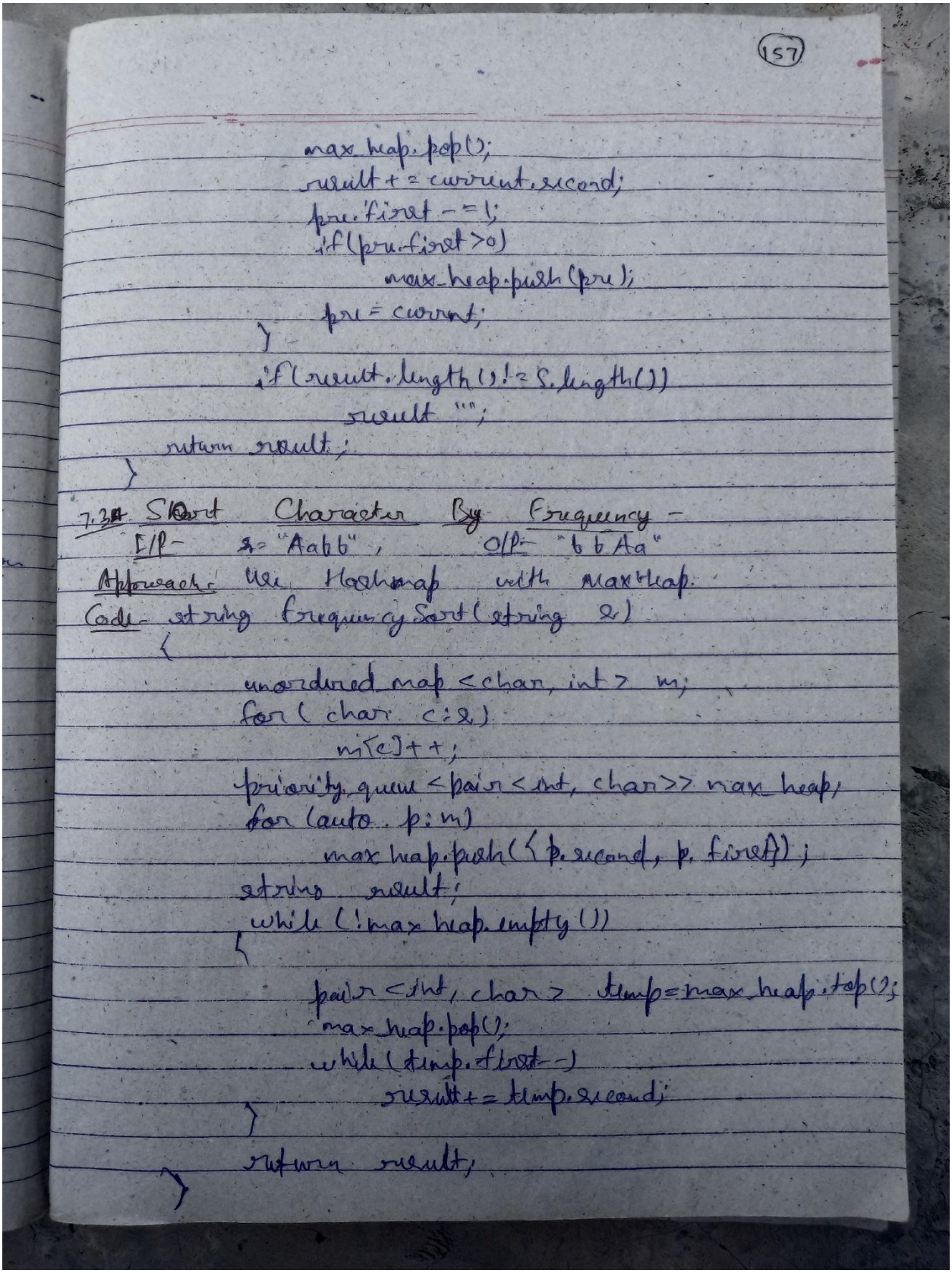


156

```
    {
        if(index >= N)
            index = 1;
        outputStr[index] = ch;
        index += 2;
    }
}
string s = "";
for(int i=0; i<N; i++)
    s += outputStr[i];
return s;
}
```

Approach-2 - Use Max Heap with frequency count of character
TC := O(N log A) (A is size of Alphabet) SC := O(N)

Code-
```
string reorganizeString(string S)
{
    if (S.empty())
        return "";
    unordered_map<char, int> m;
    for(char ch: S)
        m[ch]++;
    priority_queue<pair<int, char>> max_heap;
    for(auto it: m)
        max_heap.push({it.second, it.first});
    string result;
    pair<int, char> pre = max_heap.top();
    max_heap.pop();
    result += pre.second;
    while(!max_heap.empty())
    {
        pair<int, char> current = max_heap.top();
```



```
                max_heap.pop();
                result += current.second;
                pre.first --= 1;
                if(pre.first > 0)
                    max_heap.push(pre);
                pre = current;
            }
            if(result.length() != S.length())
                result = "";
        return result;
    }
```

7.34 Sort Character By Frequency –

I/P – s = "Aabb", O/P – "bbAa"

Approach – Use Hashmap with MaxHeap.

Code –

```
string frequencySort(string s)
{
    unordered_map<char, int> m;
    for(char c:s)
        m[c]++;
    priority_queue<pair<int, char>> max_heap;
    for(auto p:m)
        max_heap.push({p.second, p.first});
    string result;
    while(!max_heap.empty())
    {
        pair<int, char> temp = max_heap.top();
        max_heap.pop();
        while(temp.first--)
            result += temp.second;
    }
    return result;
}
```

(158)

7.35 Rotate String- T(n) = O(N)

I/P- A = "abcdi", B = "cdabd" O/P- true.

Approach- Use KMP string Matching approach -

Code- bool rotateString (string A, String B)

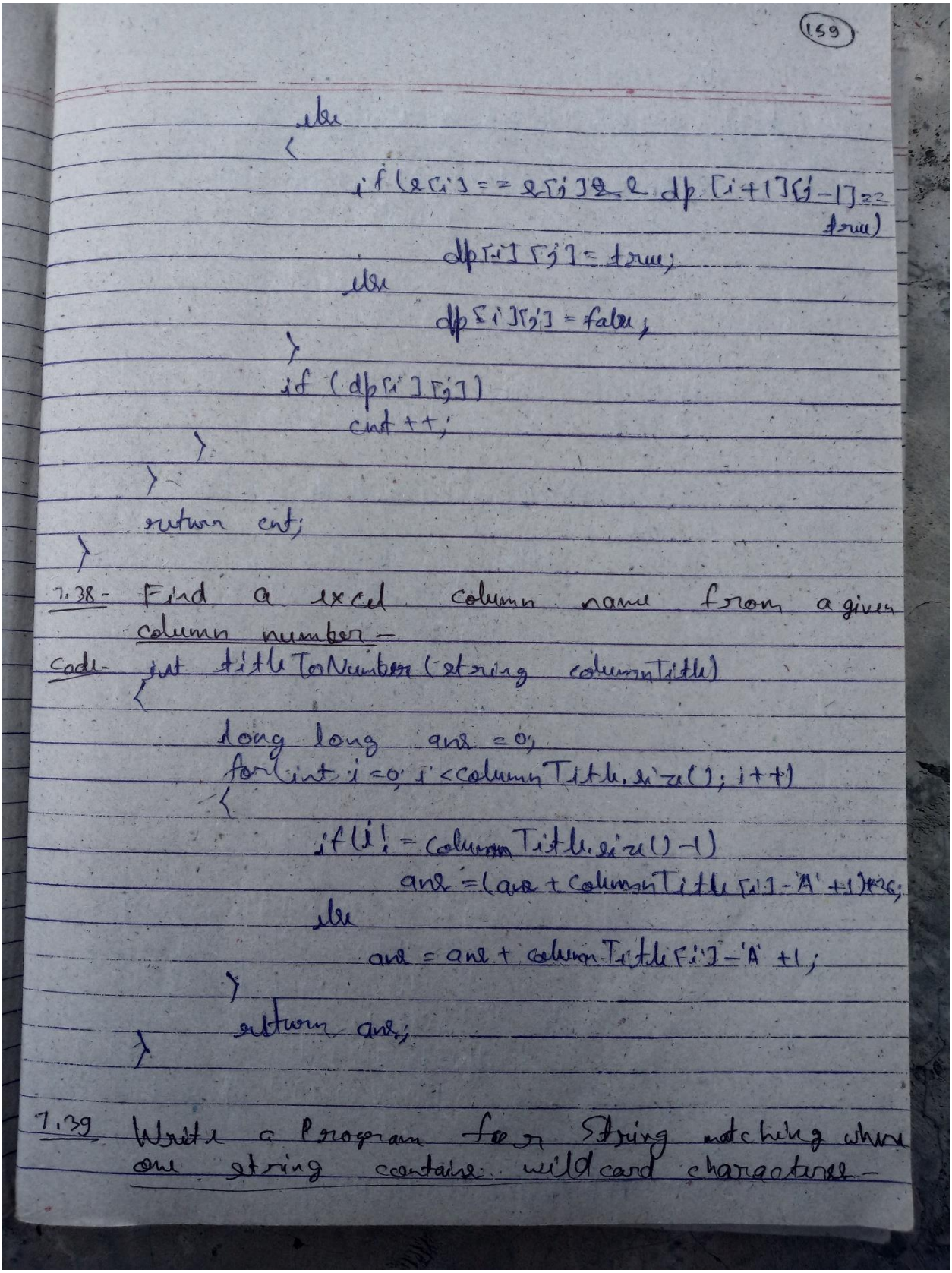
```
{
    if (A.length() != B.length())
        return false;
    string temp = A + A;
    size_t found = temp.find(B);
    if (found != string::npos)
        return true;
    return false
}
```

7.36 Find all distinct palindromic sub strings of a given string-

Approach- Use DP

Code- int countSubstrings (string s)

```
{
    int n = s.size();
    bool dp[n][n];
    int cnt = 0;
    for (int g = 0; g < n; g++)
    {
        for (int i = 0, j = g; j < n; i++, j++)
        {
            if (g == 0)
                dp[i][j] = true;
            else if (g == 1)
                dp[i][j] = s[i] == s[j];
            else
```



159

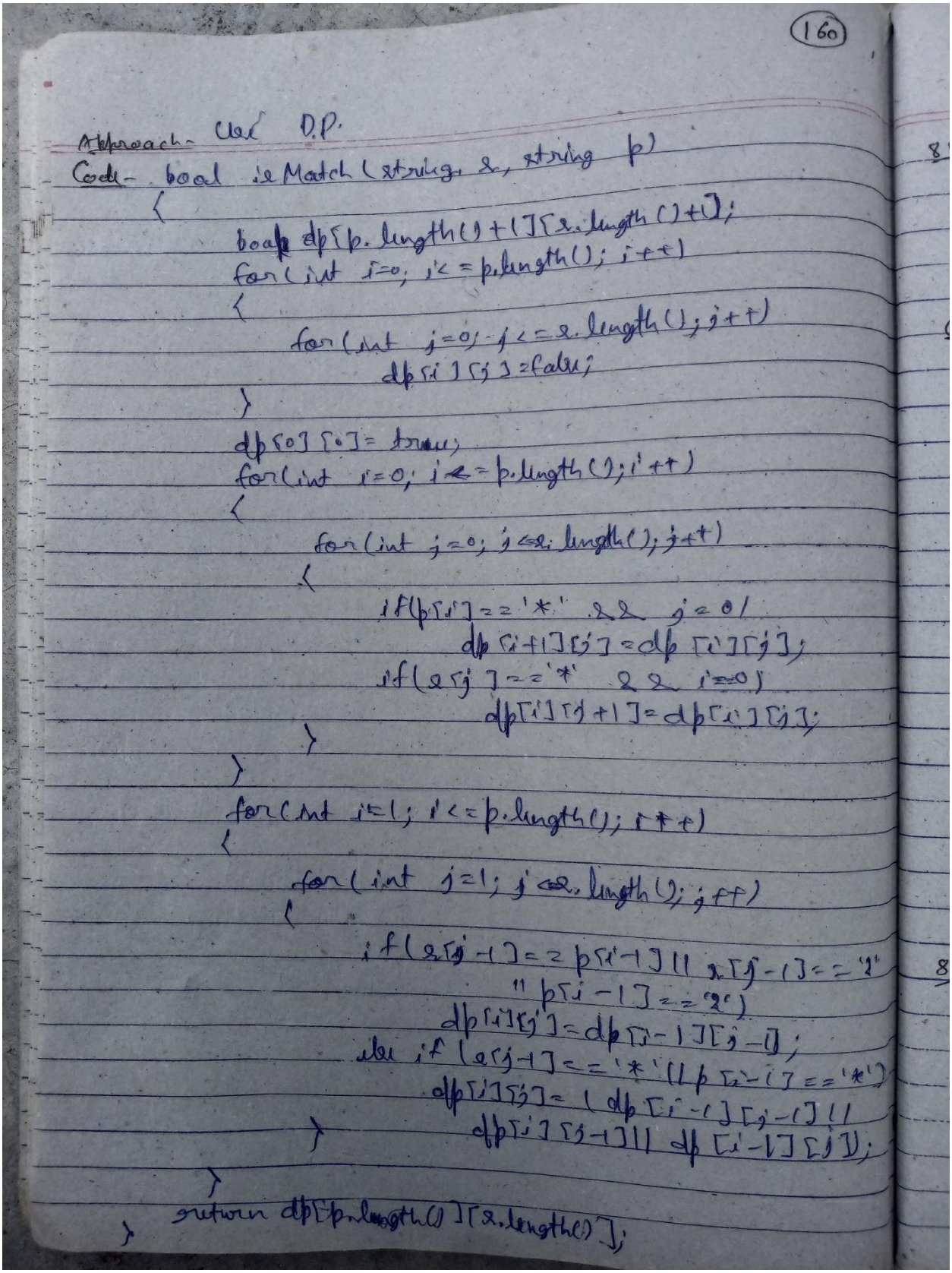
```
        else
        {
            if(s[i] == s[j] && dp[i+1][j-1] == true)
                dp[i][j] = true;
            else
                dp[i][j] = false;
        }
        if (dp[i][j])
            cnt++;
    }
}
return cnt;
}
```

7.38- Find a excel column name from a given column number -

Code-

```
int titleToNumber(string columnTitle)
{
    long long ans = 0;
    for(int i=0; i<columnTitle.size(); i++)
    {
        if(i != columnTitle.size()-1)
            ans = (ans + columnTitle[i] - 'A' + 1)*26;
        else
            ans = ans + columnTitle[i] - 'A' + 1;
    }
    return ans;
}
```

7.39 Write a Program for String matching where one string contains wildcard characters -



Approach- Use D.P.

Code-
```
bool isMatch(string s, string p)
{
    bool dp[p.length()+1][s.length()+1];
    for(int i=0; i<=p.length(); i++)
    {
        for(int j=0; j<=s.length(); j++)
            dp[i][j]=false;
    }
    dp[0][0]=true;
    for(int i=0; i<=p.length(); i++)
    {
        for(int j=0; j<=s.length(); j++)
        {
            if(p[i]=='*' && j==0)
                dp[i+1][j]=dp[i][j];
            if(s[j]=='*' && i==0)
                dp[i][j+1]=dp[i][j];
        }
    }
    for(int i=1; i<=p.length(); i++)
    {
        for(int j=1; j<=s.length(); j++)
        {
            if(s[j-1]==p[i-1] || s[j-1]=='?'
                || p[i-1]=='?')
                dp[i][j]=dp[i-1][j-1];
            else if(s[j-1]=='*'||p[i-1]=='*')
                dp[i][j]=(dp[i-1][j-1]||
                    dp[i-1][j]||dp[i][j-1]);
        }
    }
    return dp[p.length()][s.length()];
}
```

(161)

String Matching Algorithm

8.1 Naive Pattern Matching Algorithm-

I/p = "ABCDEFGH"
p = "FG"

TC is O(m,n). (n>m)

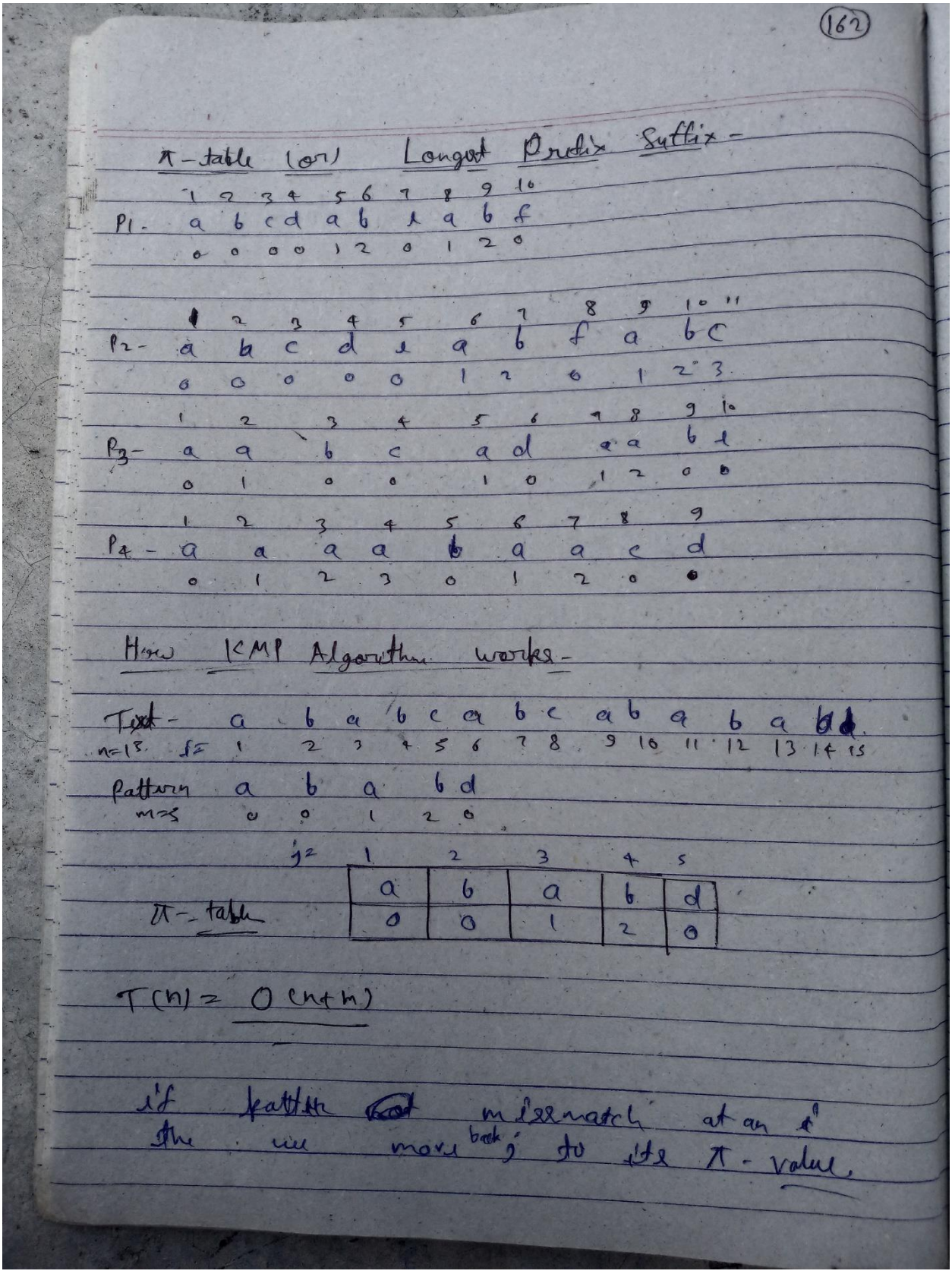
Code-

```
int match(string s1, string s2)
{
    int n = s1.length();
    int m = s2.length();
    int i, j;
    for(int i=0; i<n-m; i++)
    {
        for(j=0; j<m; j++)
        {
            if(s1[i+j] != s2[j])
                break;
        }
        if (j==m)
            return i;
    }
    return -1;
}
```

8.2 KMP (Knuth-Morris-Pratt) Algorithm

Terminology-

Pattern - a b c d a b c
Prefix - a, ab, abc, abcd, abcda ——
Suffix - c, bc, abc, dabc, ——



(162)

π-table (or) Longest Prefix Suffix -

	1	2	3	4	5	6	7	8	9	10
P1 -	a	b	c	d	a	b	a	a	b	f
	0	0	0	0	1	2	0	1	2	0

	1	2	3	4	5	6	7	8	9	10	11
P2 -	a	b	c	d	a	a	b	f	a	b	c
	0	0	0	0	0	1	2	0	1	2	3

	1	2	3	4	5	6	7	8	9	10
P3 -	a	a	b	c	a	d	a	a	b	a
	0	1	0	0	1	0	1	2	0	0

	1	2	3	4	5	6	7	8	9
P4 -	a	a	a	a	b	a	a	c	d
	0	1	2	3	0	1	2	0	0

How KMP Algorithm works -

Text - a b a b c a b c a b a b a b d

n=15 i= 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

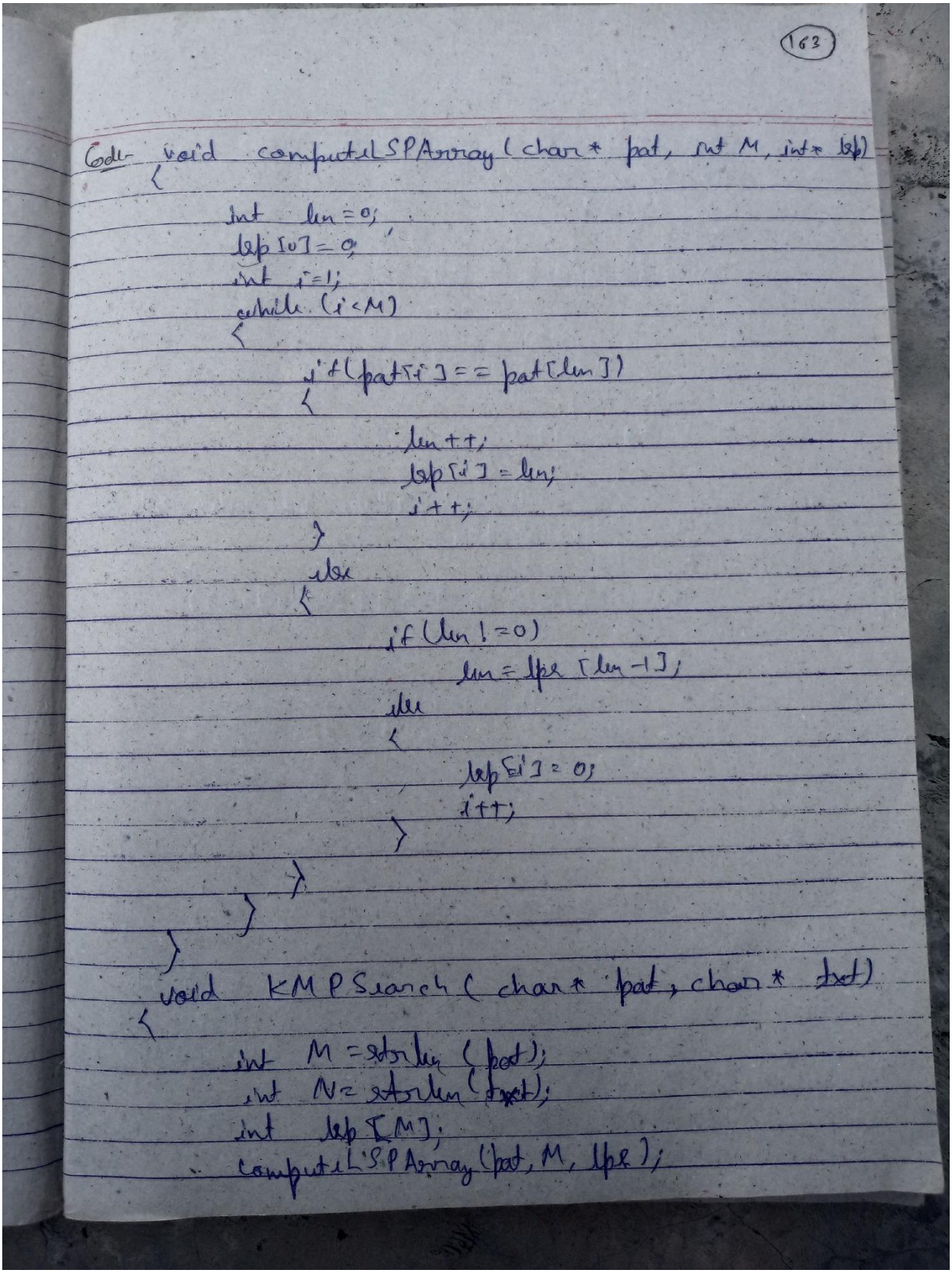
Pattern a b a b d

m=5 0 0 1 2 0

j=	1	2	3	4	5
	a	b	a	b	d
π-table	0	0	1	2	0

T(n) = O(n+m)

if pattern mismatch at any i the we move back j to its π-value.



163

Code-

```
void computeLPSArray(char* pat, int M, int* lps)
{
    int len = 0;
    lps[0] = 0;
    int i = 1;
    while (i < M)
    {
        if(pat[i] == pat[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
            if (len != 0)
                len = lps[len-1];
            else
            {
                lps[i] = 0;
                i++;
            }
        }
    }
}
void KMPSearch(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int lps[M];
    computeLPSArray(pat, M, lps);
```

(164)

```
int i=0;
int j=0;
while(i<N)
{
    if(pat[j]==txt[i])
    {
        i++;
        j++;
    }
    if(j==M)
    {
        printf("Found pattern at index %d", i-j);
        j=lps[j-1];
    }
    else if(i<N && pat[j]!=txt[i])
    {
        if(j!=0)
            j=lps[j-1];
        else
            i=i+1;
    }
}
```

8.4 Rabin Karp Algorithm

Ex- Text - a a a a a b
n=6. 1+1+2 = 4

Pattern a a b
m=3. 1+1+2 = 4

hash function — hash code

drawback — Hash collision

Single value
a → 1
b → 2
c → 3
d → 4
1 → 5

(65)

Text - c c a c c a a c d b a
n=11 4+2+1=7

Pattern d b a
m=3 4+2+1=7 hash code

Spurious Hit T(n) = O(n×m).

we create complex Hash function to reduce spurious Hit and Hash Collision.

Hash function - $P[i]\times10^{m-1} + P[i]\times10^{m-2} + \cdots$

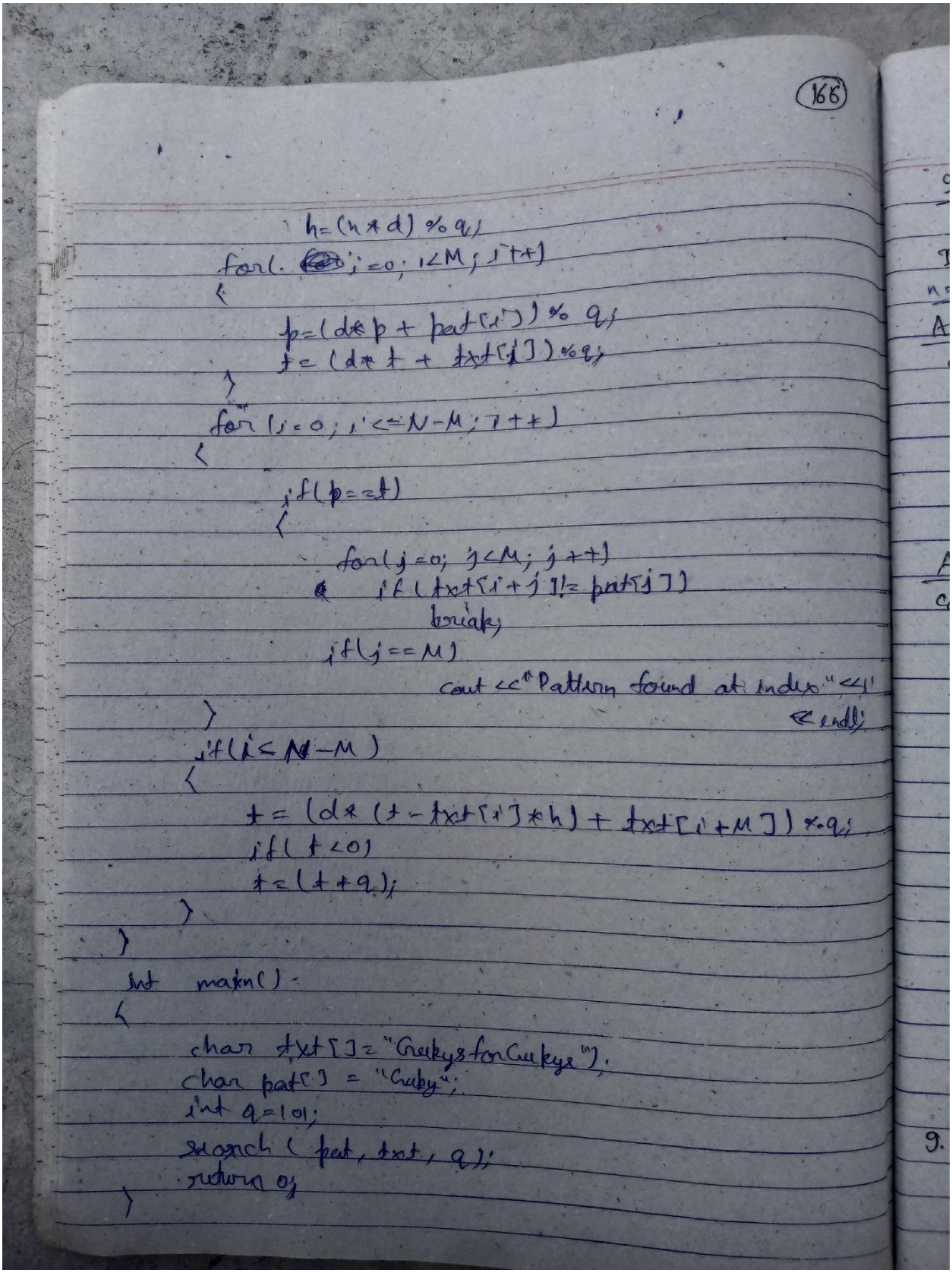
where 10 is vary according to range of character present in pattern.

Text c c a c c a a c d b a

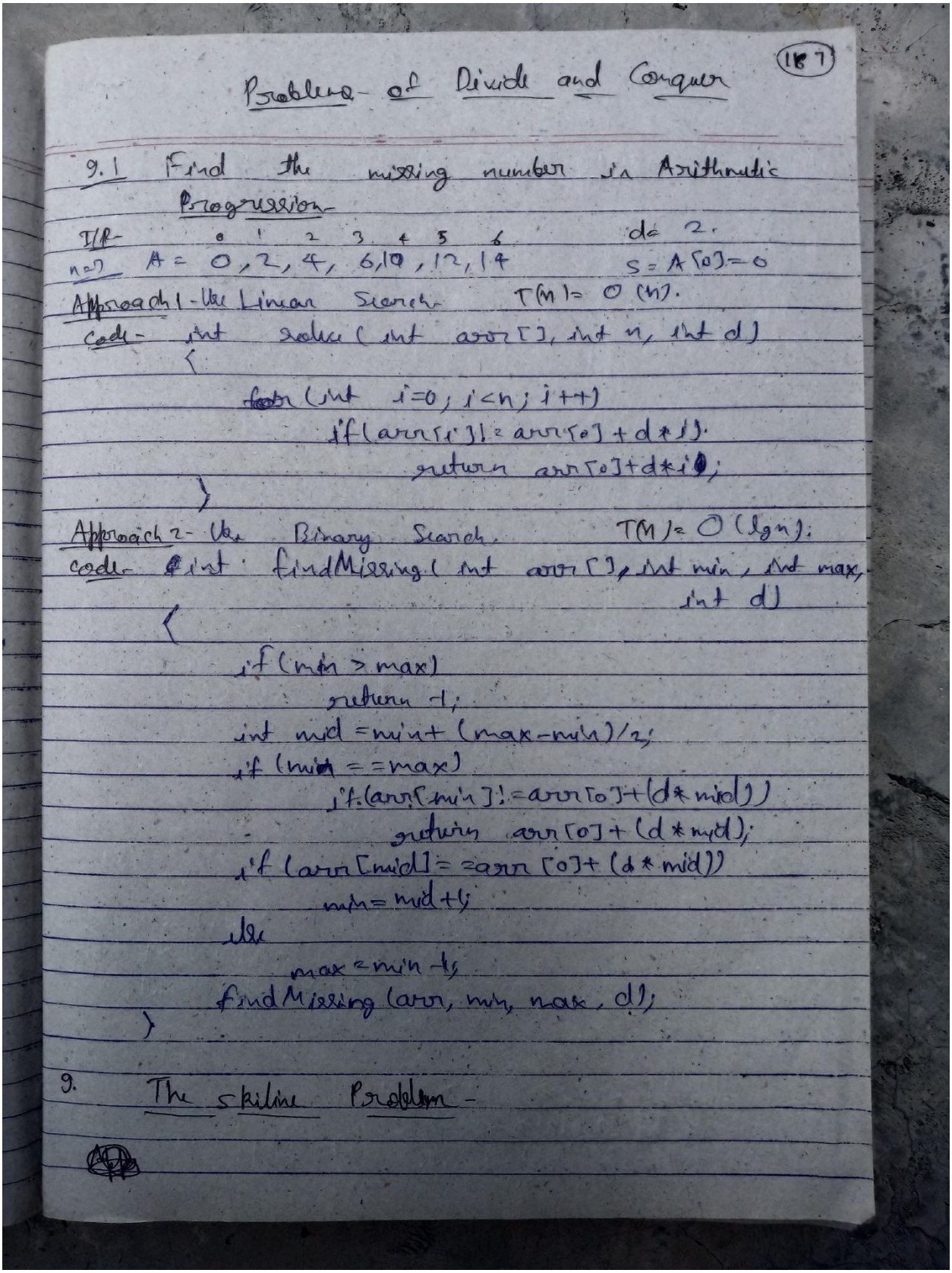
Pattern - d b a
$4\times10^2 + 2\times10^1 + 1\times10^0 = 421$

T(n) = O(n-m+1) in worst case T(n) = O(n×m)

Code -
```
#include<bits/stdc++.h>
using namespace std;
#define d 256
void search(char pat[], char txt[], int q)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int i, j;
    int p=0;
    int t=0;
    int h=1;
    for(i=0; i<M-1; i++)
```



```
        h=(h*d)%q;
    for(i=0; i<M; i++)
    {
        p=(d*p + pat[i])%q;
        t=(d*t + txt[i])%q;
    }
    for(i=0; i<=N-M; i++)
    {
        if(p==t)
        {
            for(j=0; j<M; j++)
            {
                if(txt[i+j]!=pat[j])
                    break;
            }
            if(j==M)
                cout<<"Pattern found at index "<<i
                    <<endl;
        }
        if(i<N-M)
        {
            t=(d*(t-txt[i]*h)+txt[i+M])%q;
            if(t<0)
            t=(t+q);
        }
    }
    int main()
    {
        char txt[]="Geeksforgeeks";
        char pat[]="Geeky";
        int q=101;
        search(pat, txt, q);
        return 0;
    }
```



Problems of Divide and Conquer (187)

9.1 Find the missing number in Arithmetic Progression

I/P- n=7 A = 0, 2, 4, 6, 10, 12, 14 (0 1 2 3 4 5 6) d = 2. S = A[0] = 0

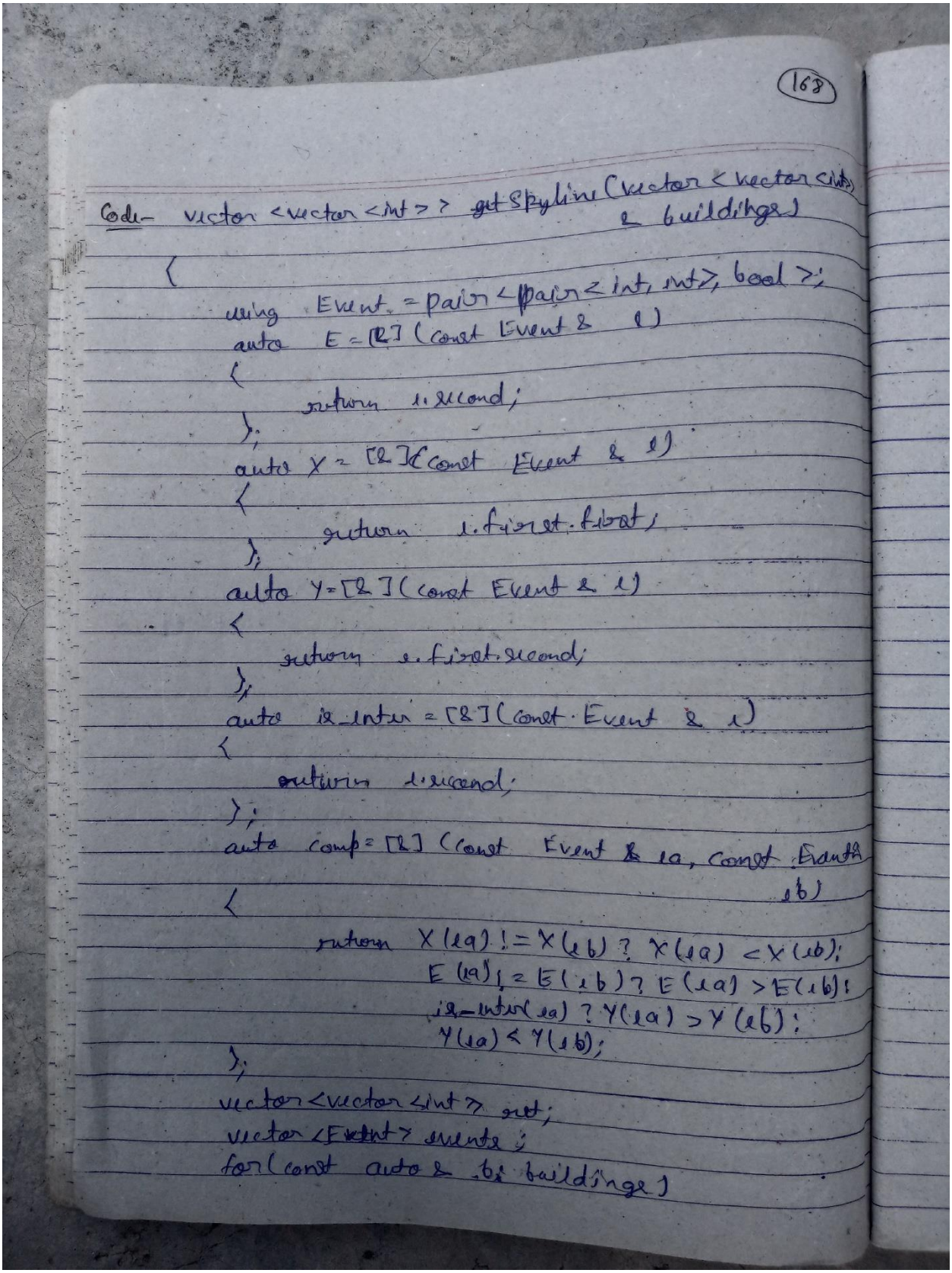
Approach 1 - Use Linear Search T(n) = O(n).

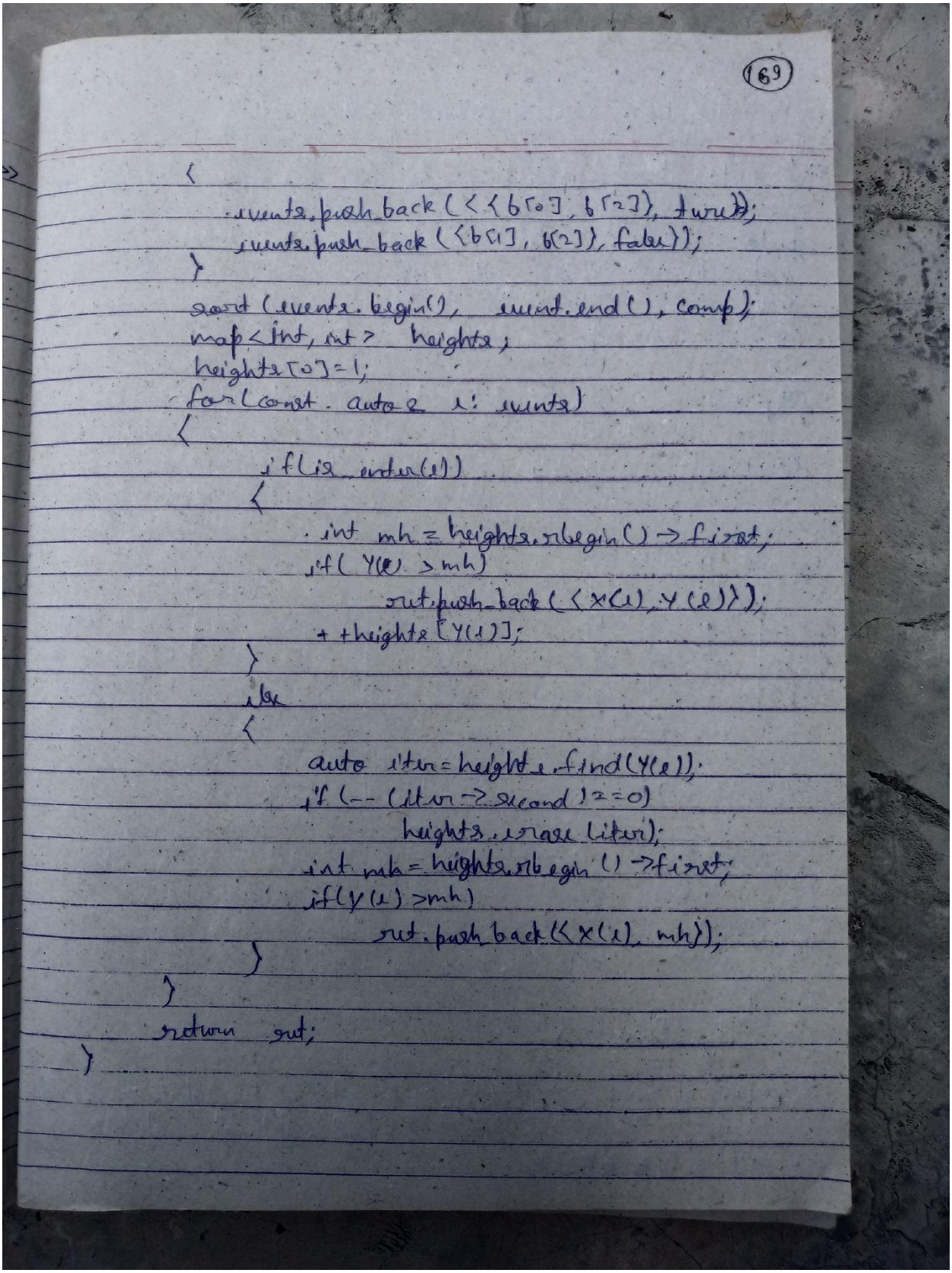
Code-
```
int solve(int arr[], int n, int d)
{
    for(int i=0; i<n; i++)
        if(arr[i] != arr[0] + d*i);
            return arr[0]+d*i;
}
```

Approach 2 - Use Binary Search. T(n) = O(lg n).

Code-
```
int findMissing(int arr[], int min, int max,
                int d)
{
    if(min > max)
        return -1;
    int mid = min + (max-min)/2;
    if(mid == max)
        if(arr[min] != arr[0]+(d*mid))
            return arr[0] + (d*mid);
    if(arr[mid] == arr[0] + (d*mid))
        min = mid+1;
    else
        max = min-1;
    findMissing(arr, min, max, d);
}
```

9. The Skiline Problem -


168
Code- vector<vector<int>> getSkyline(vector<vector<int>>& buildings)
{
using Event = pair<pair<int,int>, bool>;
auto E = [&](const Event& e)
{
return e.second;
};
auto X = [&](const Event& e)
{
return e.first.first;
};
auto Y = [&](const Event& e)
{
return e.first.second;
};
auto is_entry = [&](const Event& e)
{
return e.second;
};
auto comp = [&](const Event& ea, const Event& eb)
{
return X(ea) != X(eb) ? X(ea) < X(eb):
E(ea) != E(eb) ? E(ea) > E(eb):
is_entry(ea) ? Y(ea) > Y(eb):
Y(ea) < Y(eb);
};
vector<vector<int>> ret;
vector<Event> events;
for(const auto& bi: buildings)



```
(69)

    {
        events.push_back({{b[0], b[2]}, true});
        events.push_back({{b[1], b[2]}, false});
    }
    sort(events.begin(), event.end(), comp);
    map<int, int> heights;
    heights[0]=1;
    for(const auto& i: events)
    {
        if(is_enter(i))
        {
            int mh = heights.rbegin()->first;
            if( Y(i) > mh)
                ret.push_back({X(i), Y(i)});
            ++heights[Y(i)];
        }
        else
        {
            auto iter=height_s.find(Y(i));
            if(--(iter->second)==0)
                heights.erase(iter);
            int mh=heights.rbegin()->first;
            if(Y(i)>mh)
                ret.push_back({X(i), mh});
        }
    }
    return ret;
}
```

(170)

Problems on Greedy Algorithms

(i) Fractional Knapsack - Given weights and values of n items, we need to put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

Terminology (optimization) -

Objective - max (total value of items in knapsack) such that

Constraints - (i) We can pick fractions of any item
(ii) the total weight of items we pick $\leq W$

Greedy Approach -

Items	val($)	wt (Kg)	val/wt.		
Rice 1	60	10	60/10=6	80	20kg ③
Wheat 2	100	20	5	100	20kg ②
Milk 3	120	30	4	60	10kg ①
	W = 50 Kg			240	Soln

Time complexity fractional knapsack -

$T(n) = O(n)$ (i) val/wt for each item → O(n) time, O(n) space
$S(n) = O(n)$ (ii) Sort by val/wt → O(n lg n) time.
(iii) Start picking greedily → O(n)
till the knapsack is full.

Greedy Strategy -

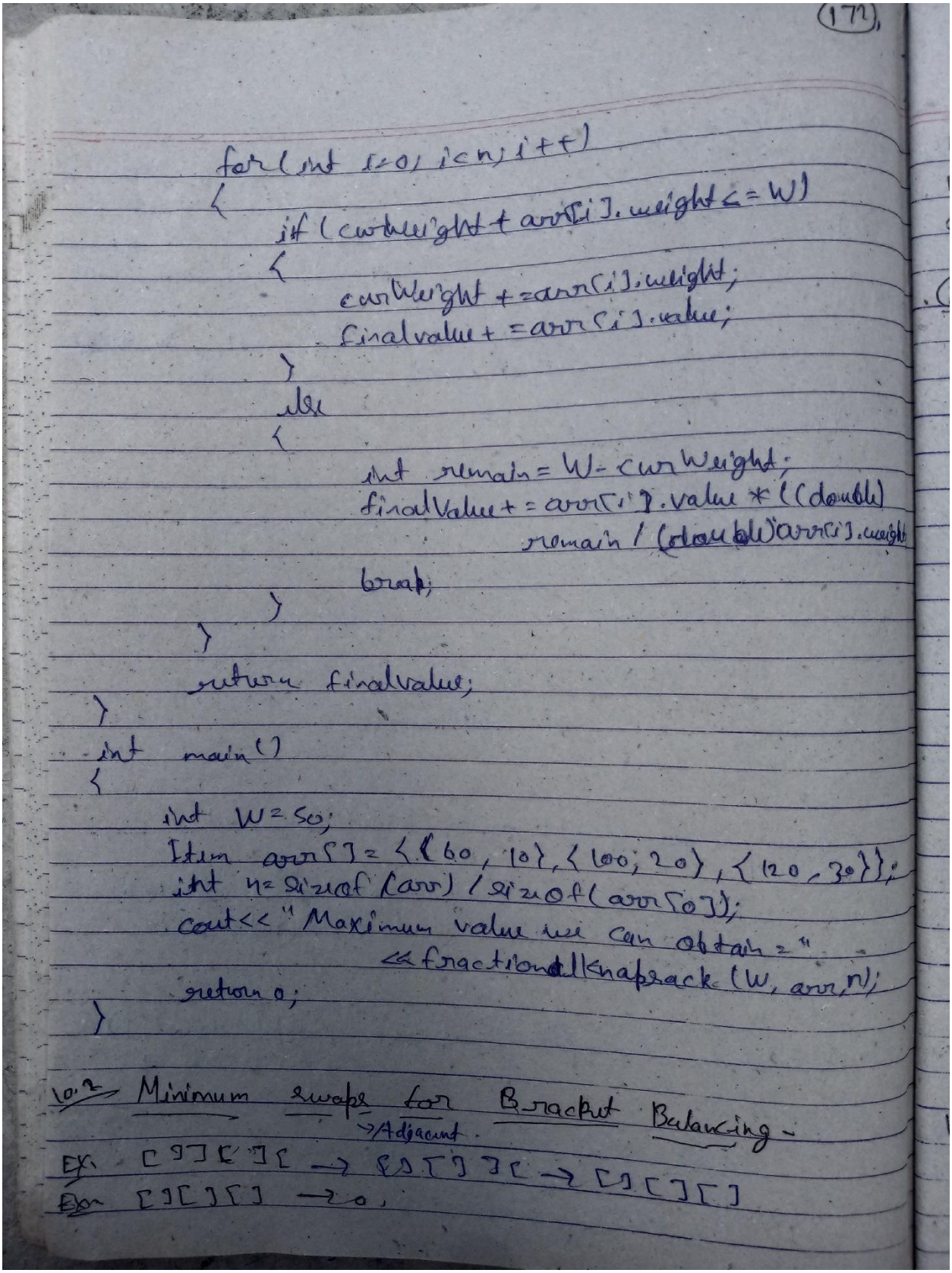
(i) Optimal Sub Structure - an optimal solution will contain optimal solution of sub-problem. (Recursion).

(171)

iii) Greedy Choice Property - at every step, if we greedily pick the solution ⇒ eventually resulting in an optimal (globally optimal solution) but (locally optimal solution)

Note- Greedy Choice property need not be satisfied for all real world problem.

Code- #include<bits/stdc++.h>
using namespace std;
struct Item{
int value, weight;
Item(int value, int weight)
{
this->value=value;
this->weight=weight;
}
};
bool cmp(struct Item a, struct Item b)
{
double r1=(double)a.value/(double)a.weight;
double r2=(double)b.value/(double)b.weight;
return r1>r2;
}
double fractionalKnapsack(int W, struct Item arr[], int n)
{
sort(arr, arr+n, cmp);
int curWeight=0;
double finalvalue=0.0;



(172)

```
for(int i=0; i<n; i++)
{
    if(curWeight + arr[i].weight <= W)
    {
        curWeight += arr[i].weight;
        finalvalue += arr[i].value;
    }
    else
    {
        int remain = W - curWeight;
        finalValue += arr[i].value * ((double)
                      remain / (double)arr[i].weight
        break;
    }
}
return finalvalue;
}
int main()
{
    int W = 50;
    Item arr[] = {{60, 10}, {100, 20}, {120, 30}};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Maximum value we can obtain = "
         << fractionalKnapsack(W, arr, n);
    return 0;
}
```

10.2 → Minimum swaps for Bracket Balancing - → Adjacent.

Ex: [ ]] [ ] [→ [ ] [ ] [ ] → [ ] [ ] [ ]

Ex [ ] [ ] [ ] → 0.

(73)

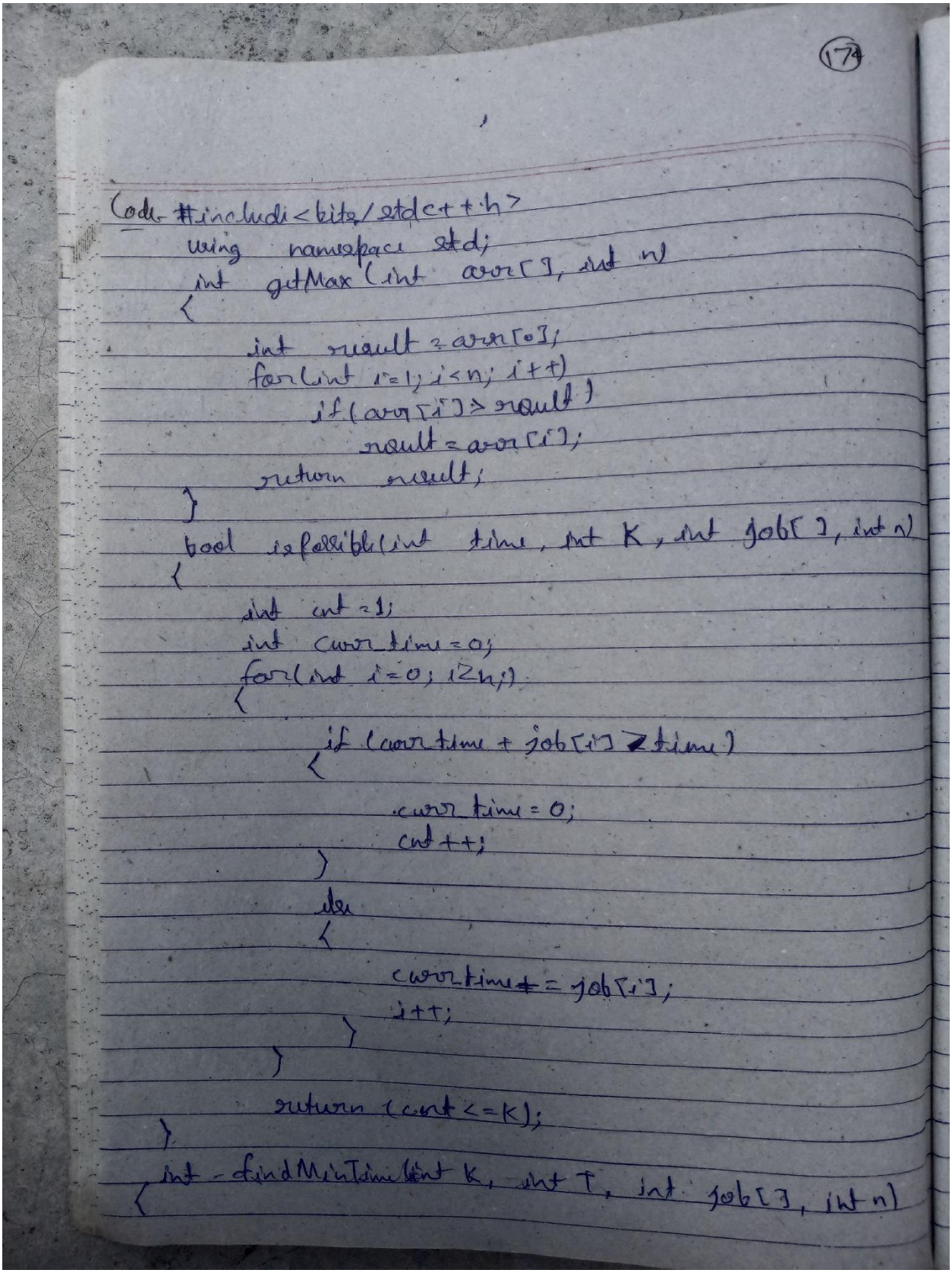
Ex [[[ ]]][→ [[[ ]]][]

Ex][[→][][→ [][][→ [][][]

Approach - Greedy Approach - T(n) = O(n) Space = O(1)

```
Code - int minimumNumberOfSwaps(string S)
{
    int countL=0, countR=0, swaps=0, imbalance=0;
    for(int i=0; i<S.length(); i++)
    {
        if(S[i]=='[')
        {
            countL++;
            if(imbalance>0)
            {
                swaps+=imbalance;
                imbalance--;
            }
        }
        else if(S[i]==']')
        {
            countR++;
            imbalance=countR-countL;
        }
    }
    return swaps;
}
```

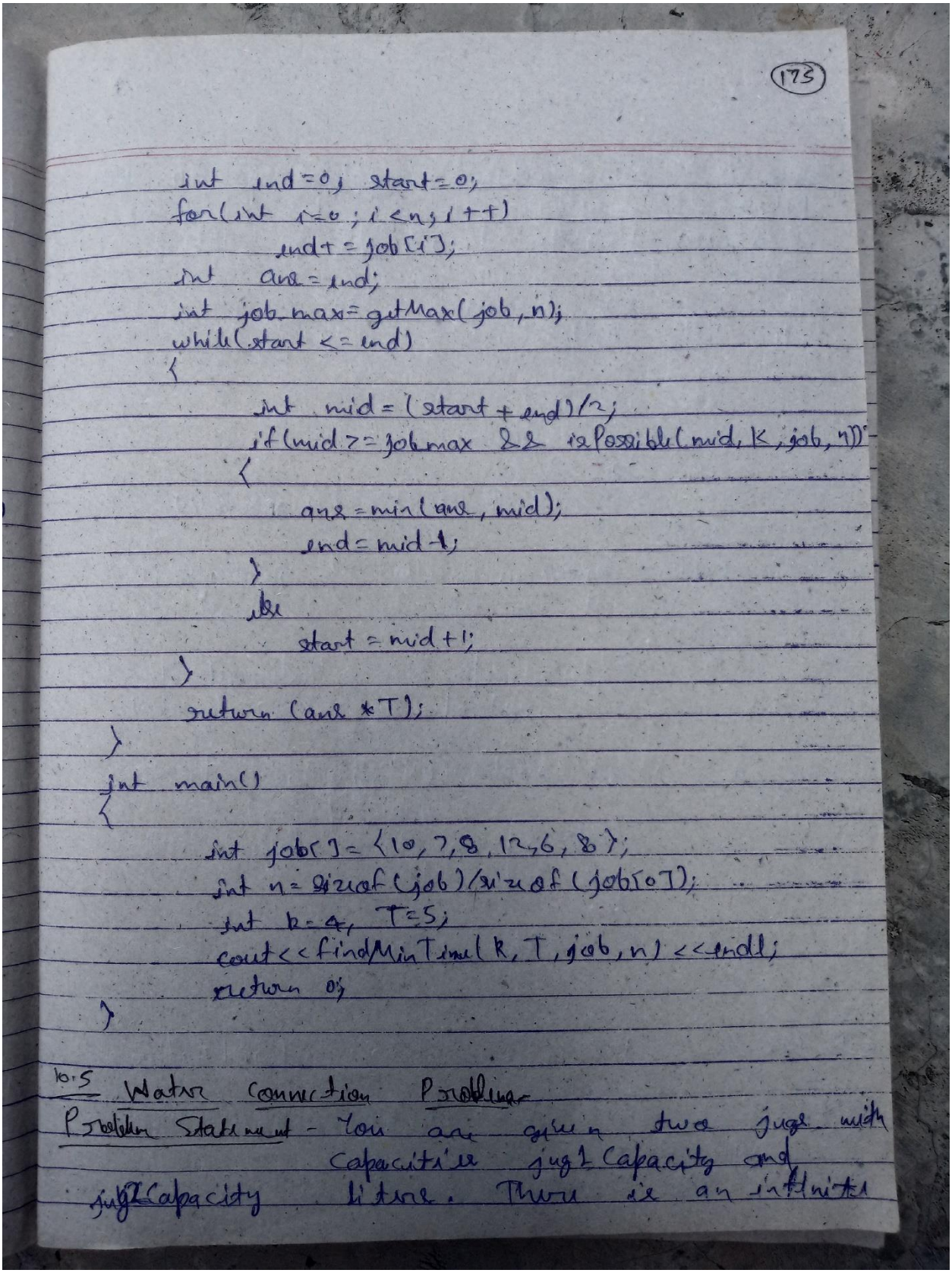
10.3 Given an array of jobs with different time intervals. Find the minimum time to finish all jobs -



(174)

Code:

```
#include<bits/stdc++.h>
using namespace std;
int getMax(int arr[], int n)
{
    int result = arr[0];
    for(int i=1; i<n; i++)
        if(arr[i] > result)
            result = arr[i];
    return result;
}
bool isFeasible(int time, int K, int job[], int n)
{
    int cnt = 1;
    int curr_time = 0;
    for(int i=0; i<n;)
    {
        if(curr_time + job[i] > time)
        {
            curr_time = 0;
            cnt++;
        }
        else
        {
            curr_time += job[i];
            i++;
        }
    }
    return (cnt <= K);
}
int findMinTime(int K, int T, int job[], int n)
{
```



(175)

```
int end=0, start=0;
for(int i=0; i<n; i++)
    end+=job[i];
int ans=end;
int job_max=getMax(job,n);
while(start<=end)
{
    int mid=(start+end)/2;
    if(mid>=job_max && isPossible(mid,K,job,n))
    {
        ans=min(ans,mid);
        end=mid-1;
    }
    else
        start=mid+1;
}
return (ans*T);
}
int main()
{
    int job[]={10,7,8,12,6,8};
    int n=sizeof(job)/sizeof(job[0]);
    int k=4, T=5;
    cout<<findMinTime(k,T,job,n)<<endl;
    return 0;
}
```

10.5 Water connection Problem

Problem Statement - You are given two jugs with capacities jug1 Capacity and jug2 Capacity litres. There is an infinite

(176)

amount of water supply available. Determine whether it is possible to measure exactly targetCapacity liters using these two jugs.

Approach: Use Greedy Approach.

Code-

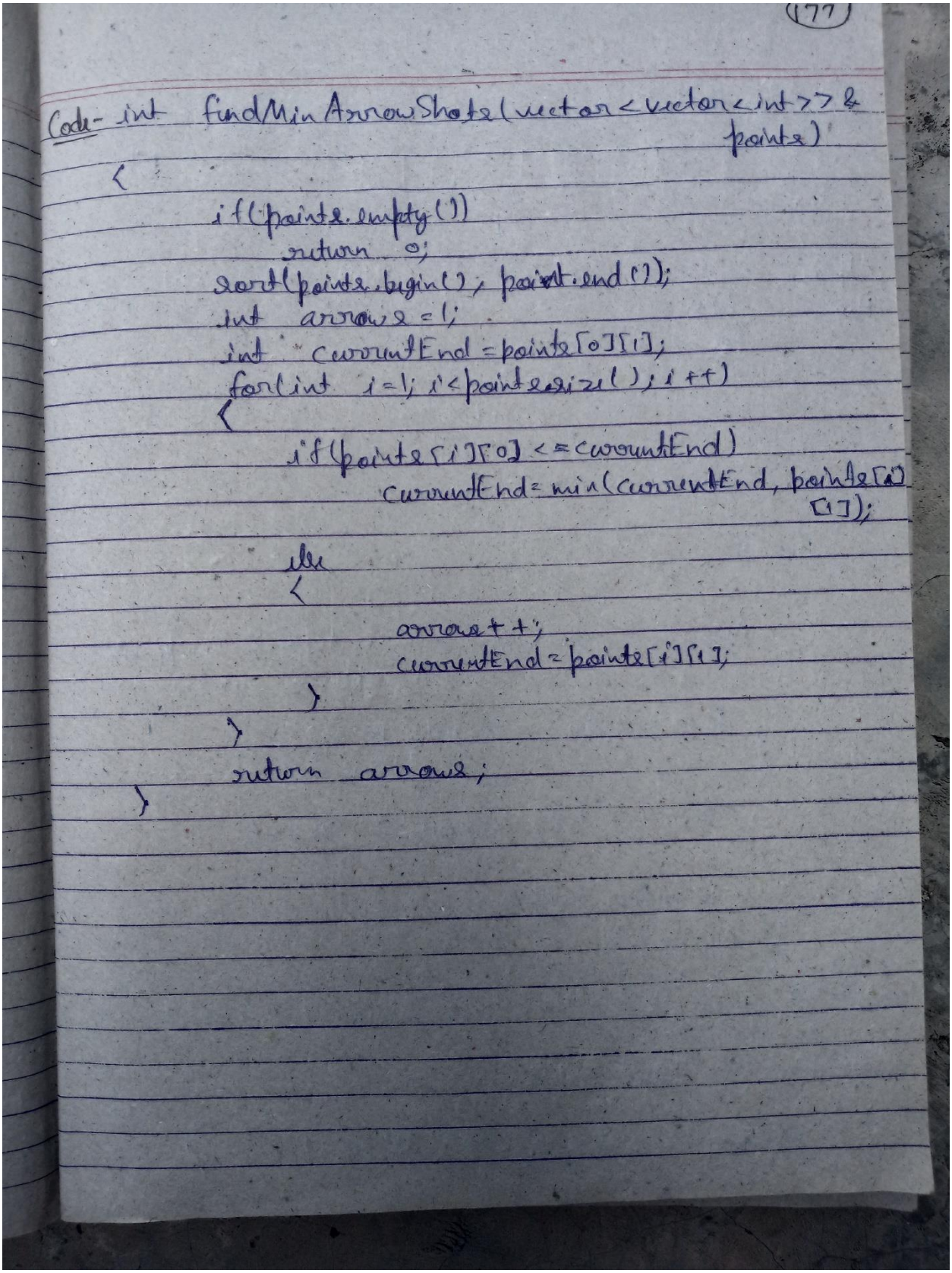
```
int helper(int x, int y)
{
    if(x%y == 0)
        return y;
    else
        return helper(y, x%y);
}
bool canMeasureWater(int jug1Capacity,
    int jug2Capacity, int targetCapacity)
{
    if(targetCapacity < 0 || targetCapacity >
        jug1Capacity + jug2Capacity)
        return false;
    int minPossible = helper(jug1Capacity,
                    jug2Capacity);
    if(targetCapacity % minPossible == 0)
        return true;
    else
        return false;
}
```

10.8 Minimum Number of Arrow to Burst Balloons-

Ex: I/P Points = [[1,2],[3,4],[5,6],[7,8]]

O/P 4

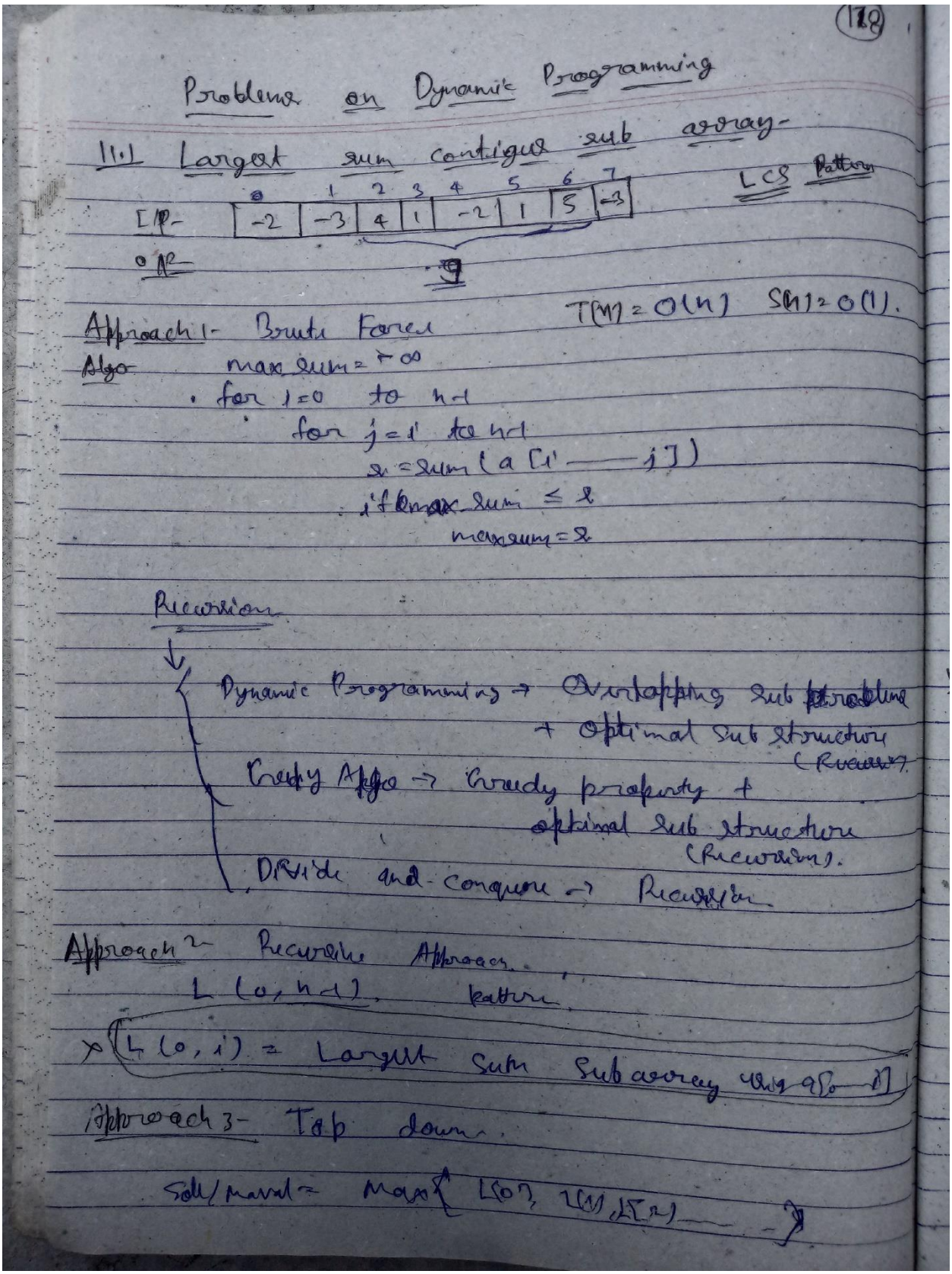
Approach: Use Greedy Approach-



(77)

Code- int findMinArrowShots(vector<vector<int>>& points)

```
{
    if(points.empty())
        return 0;
    sort(points.begin(), points.end());
    int arrows = 1;
    int currentEnd = points[0][1];
    for(int i=1; i<points.size(); i++)
    {
        if(points[i][0] <= currentEnd)
            currentEnd = min(currentEnd, points[i][1]);
        else
        {
            arrows++;
            currentEnd = points[i][1];
        }
    }
    return arrows;
}
```



Approach 3- Top down.

Sol(n) = Max{ L[0], L[1], L[2] ---- }

(79)

Approach-4 - Bottom up (Iterative) Approach. $T(n) = O(n)$
$S(n) = O(1)$

Code -
```
int maxSubArray(vector<int>& nums)
{
    int max_sum = nums[0];
    int max_curr = nums[0];
    for(int i=1; i<nums.size(); i++)
    {
        max_curr = max(nums[i], max_curr+nums[i]);
        if(max_sum < max_curr)
            max_sum = max_curr;
    }
    return max_sum;
}
```

11.2 Longest Palindromic sub sequence -

→ ABA ← non-contiguous.

Ex- ABCBCA → ABBA → ↑ length
ACCA
ABA

Need not be unique.

Approach 1- Brute force $T(n) = O(n \cdot 2^n)$

1) generate all sub sequences. (2^n)
2) Retain only palindromic sub sequences. (n).
3) Pick the length of the longest Palindromic sub sequence (LPS).

(180)

Approach2- Use Recursion. $T(n) = O(n^2)$

$$L(i,j) = \begin{cases} 1 & \text{if } i=j \\ \text{not defined} & \text{if } i>j \\ L(i+1, j-1)+2 & \text{if } a[i] = a[j] \\ \max(L(i, j+1), L(i+1, j)) & \text{otherwise} \end{cases}$$

Approach-3- DP (Memoization). $T(n) = O(n^2)$.

recursion + matrix $L[0 \ldots n-1][0 \ldots n-1]$.

Approach 4- DP (Bottom up). $T(n) = O(n^2)$ $S(n) = O(n^2)$

Code- int longestPalindromeSubseq (string s)

{

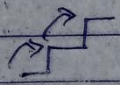
int n = s.length();

int LPS[n+1][n+1];

for (int i=0; i<n; i++)

```
LPS[i][i]=1;
for(int k=2; k<=n; k++)
{
    for(int i=0; i<=n-k; i++)
    {
        int j=i+k-1;
        if(s[i]==s[j] && k==2)
            LPS[i][j]=2;
        else if(s[i]==s[j])
            LPS[i][j]=LPS[i+1][j-1]+2;
        else
            LPS[i][j]=max(LPS[i][j-1],
                          LPS[i+1][j]);
    }
}
return LPS[0][n-1];
}
```

11.3 Climbing stairs Problem.

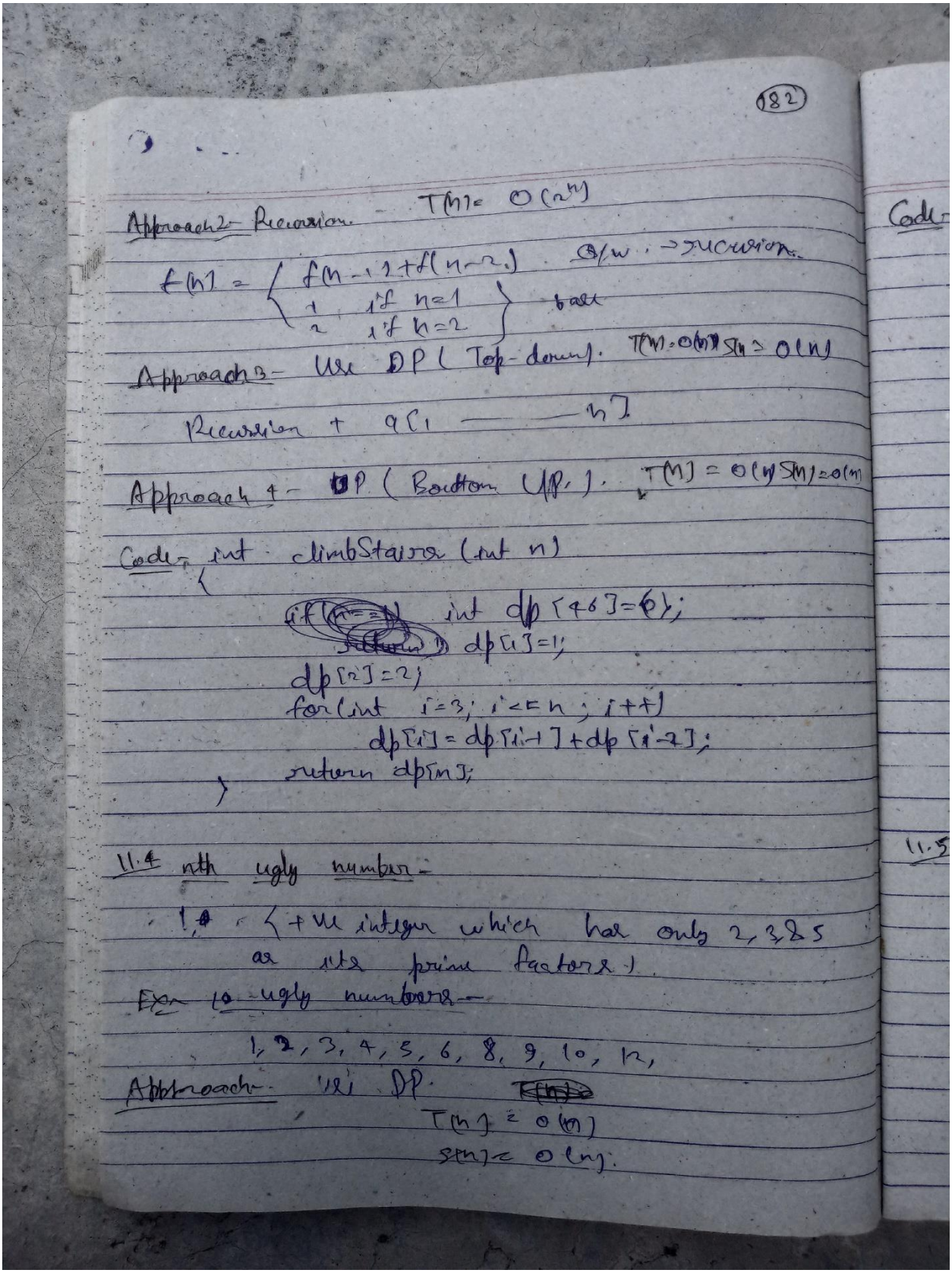


n stairs → 1 stair
→ 2 stairs

Find No. of ways to reach top.

Ex:- n = 4

1, 1, 1, 1
2, 1, 1
1, 2, 1
1, 1, 2
2, 2
...



182

Approach 2- Recursion. $T[n] = O(2^n)$

$f[n] = \begin{cases} f(n-1)+f(n-2) & \text{O/w} \rightarrow \text{recursion} \\ 1 & \text{if } n=1 \\ 2 & \text{if } n=2 \end{cases}$ base

Approach 3- Use DP (Top-down). $T[n] = O(n)$ $S[n] = O(n)$

Recursion + a[1 ---- n]

Approach 4- DP (Bottom Up). $T[n] = O(n)$ $S[n] = O(n)$

Code- int climbStairs (int n)

{

if(n==1) int dp[n+6]={0};

return 1; dp[1]=1;

dp[2]=2;

for(int i=3; i<=n; i++)

dp[i] = dp[i-1] + dp[i-2];

return dp[n];

}

11.4 nth ugly number-

Def- {+ve integer which has only 2, 3 & 5 as its prime factors}.

Ex- 10 ugly numbers-

1, 2, 3, 4, 5, 6, 8, 9, 10, 12,

Approach- Use DP. T[n] = O(n)

S[n] = O(n).

11.5

Code-

(183)

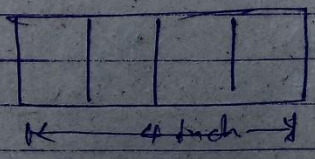
Code-

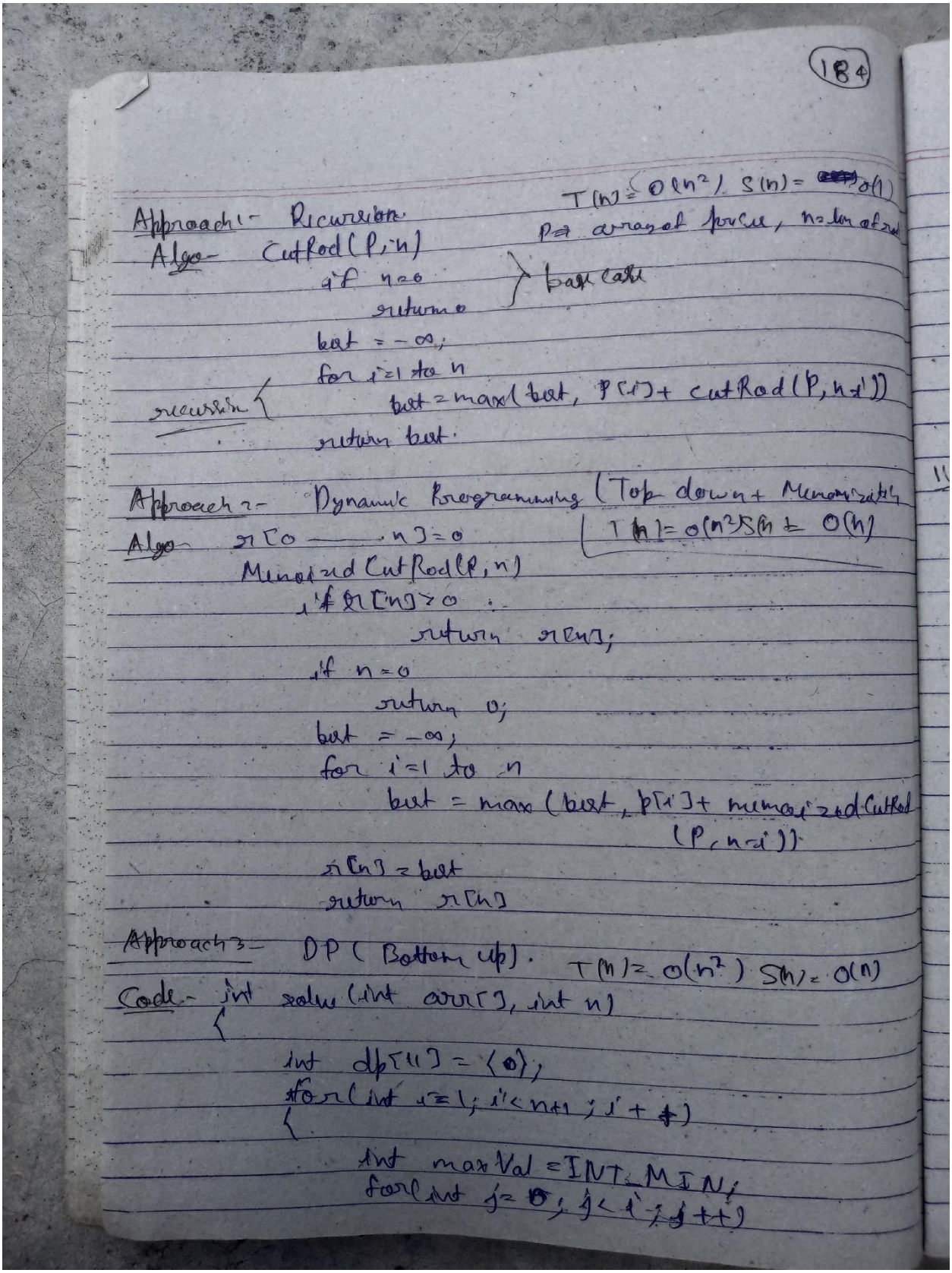
```
int nthUglyNumber(int n)
{
    int dp[1690]={0};
    int p2=0, p3=0, p5=0;
    dp[0]=1;
    for(int i=0; i<n; i++)
    {
        int next=min(dp[p2]*2, dp[p3]*3);
        next = min(next, dp[p5]*5);
        dp[i] = next;
        if(next==dp[p2]*2)
            p2++;
        if(next==dp[p3]*3)
            p3++;
        if(next==dp[p5]*5)
            p5++;
    }
    return dp[n-1];
}
```

11.5 Rod-cutting Problem-

len	1	2	3	4	5	6	7	8	9	10
Price	1	5	8	9	10	17	17	20	24	30

n = len. = 4.



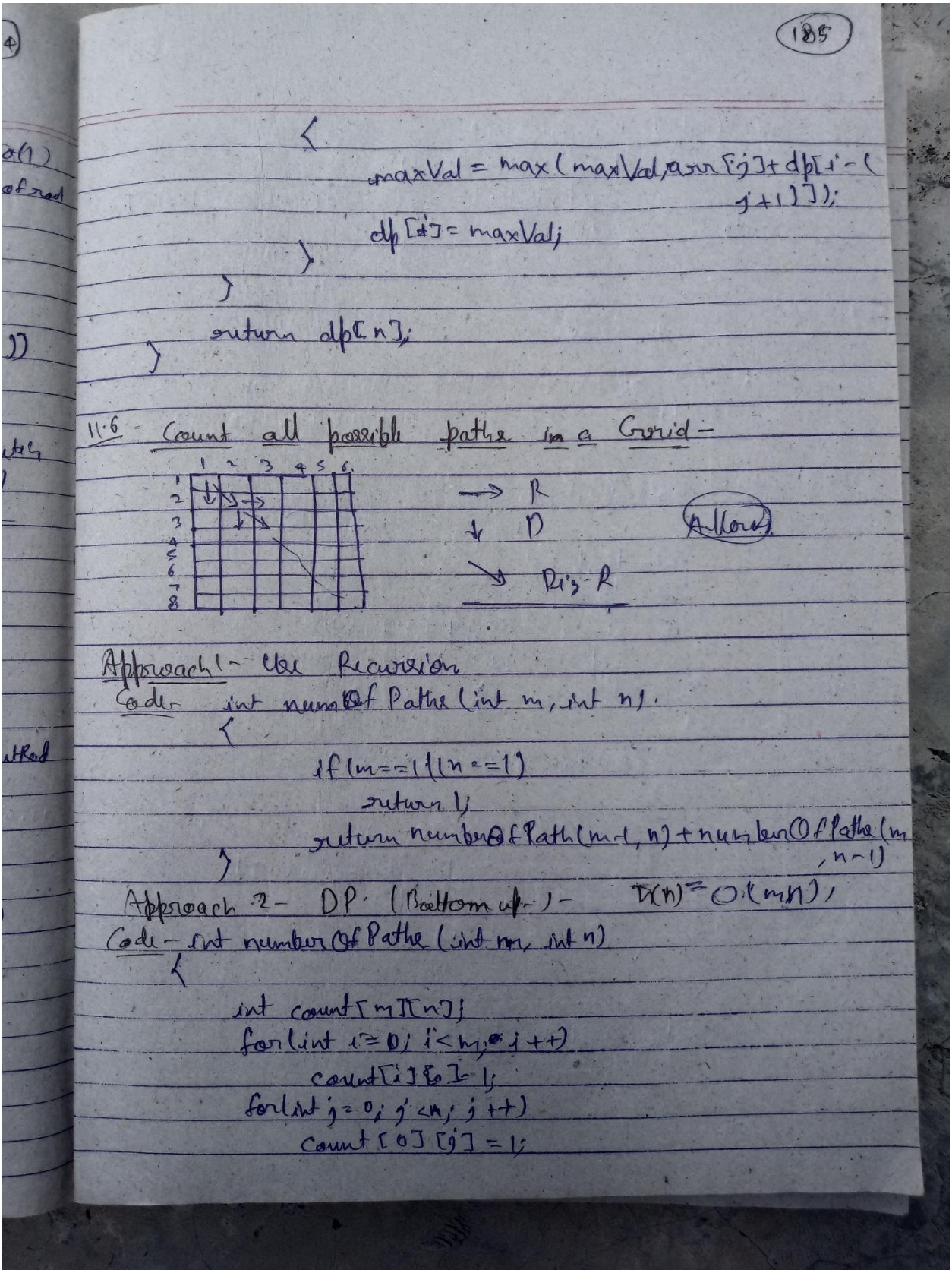


184

Approach:- Recursion. T(n) = O(n²) / S(n) = O(1)
Algo- CutRod(P, n) P → array of price, n → length of rod
if n=0
return 0 } base case
best = -∞;
for i=1 to n
best = max(best, P[i] + CutRod(P, n-i)) } recursion
return best.

Approach 2:- Dynamic Programming (Top down + Memoization) T(n) = O(n²) S(n) = O(n)
Algo- r[0 ..... n] = 0
MemoizedCutRod(P, n)
if r[n] ≥ 0
return r[n];
if n=0
return 0;
best = -∞;
for i=1 to n
best = max(best, p[i] + memoizedCutRod(P, n-i))
r[n] = best
return r[n]

Approach 3:- DP (Bottom up). T(n) = O(n²) S(n) = O(n)
Code- int solve(int arr[], int n)
{
int dp[n+1] = {0};
for(int i=1; i<n+1; i++)
{
int maxVal = INT_MIN;
for(int j=0; j<i; j++)



185

```
        {
            maxVal = max(maxVal, arr[j] + dp[i - (j+1)]);
        }
        dp[i] = maxVal;
    }
    return dp[n];
}
```

11.6 Count all possible paths in a Grid -

→ R
↓ D
↘ Ri - R

Allowed

Approach 1 - Use Recursion

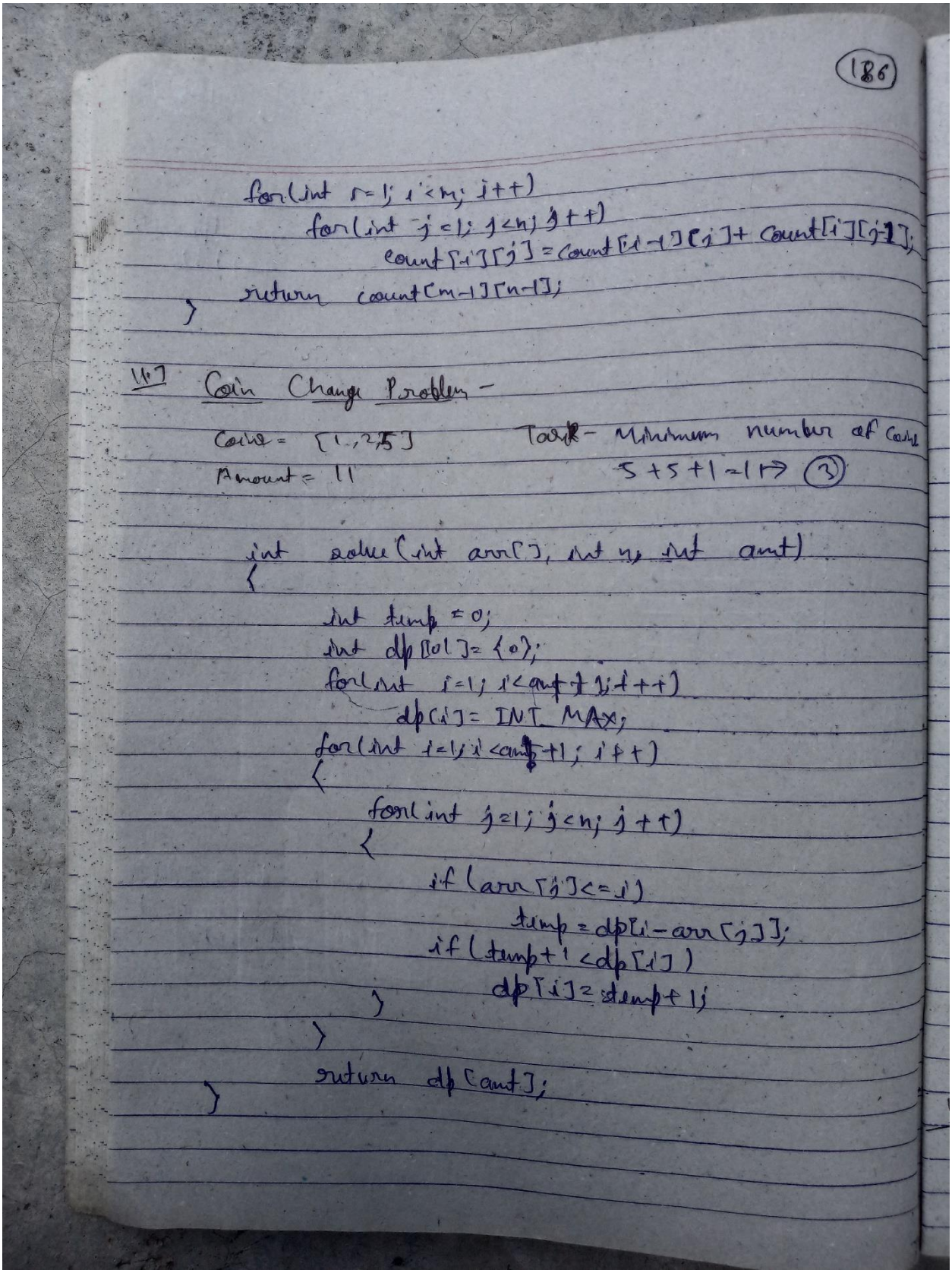
Code -

```
int numberOfPaths(int m, int n){
    if(m==1||(n==1))
        return 1;
    return numberOfPath(m-1, n) + numberOfPaths(m, n-1)
}
```

Approach 2 - DP. (Bottom up -) - T(n) = O(m,n),

Code -

```
int numberOfPaths(int m, int n)
{
    int count[m][n];
    for(int i=0; i<m; i++)
        count[i][0] = 1;
    for(int j=0; j<n; j++)
        count[0][j] = 1;
```



186

```
for(int i=1; i<m; i++)
    for(int j=1; j<n; j++)
        count[i][j] = count[i-1][j] + count[i][j-1];
return count[m-1][n-1];
}
```

11.7 Coin Change Problem -

Coins = [1, 2, 5]
Amount = 11

Task - Minimum number of coins
5 + 5 + 1 = 11 ⇒ (3)

```
int solve(int arr[], int n, int amt)
{
    int temp = 0;
    int dp[101] = {0};
    for(int i=1; i<amt+1; i++)
        dp[i] = INT_MAX;
    for(int i=1; i<amt+1; i++)
    {
        for(int j=1; j<n; j++)
        {
            if(arr[j]<=i)
                temp = dp[i-arr[j]];
            if(temp+1 < dp[i])
                dp[i] = temp+1;
        }
    }
    return dp[amt];
}
```

187

11.8 Minimum Cost path Problem-

2	1	5	1
3	4	2	2
1	2	3	3
1	3	2	4

Original Matrix.

2	3	8	9
5	7	9	11
6	8	11	14
7	10	12	16

Cost matrix.

Approach: Use Recursion.

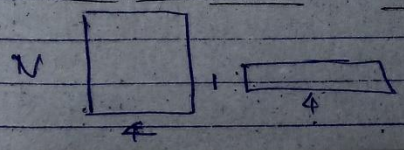
$$C[i][j] = a[i][j] + \min(C[i][j-1], C[i-1][j])$$

Approach 2: Use Bottom up DP. $T(n) = O(mn)$ $S(n) = O(mn)$

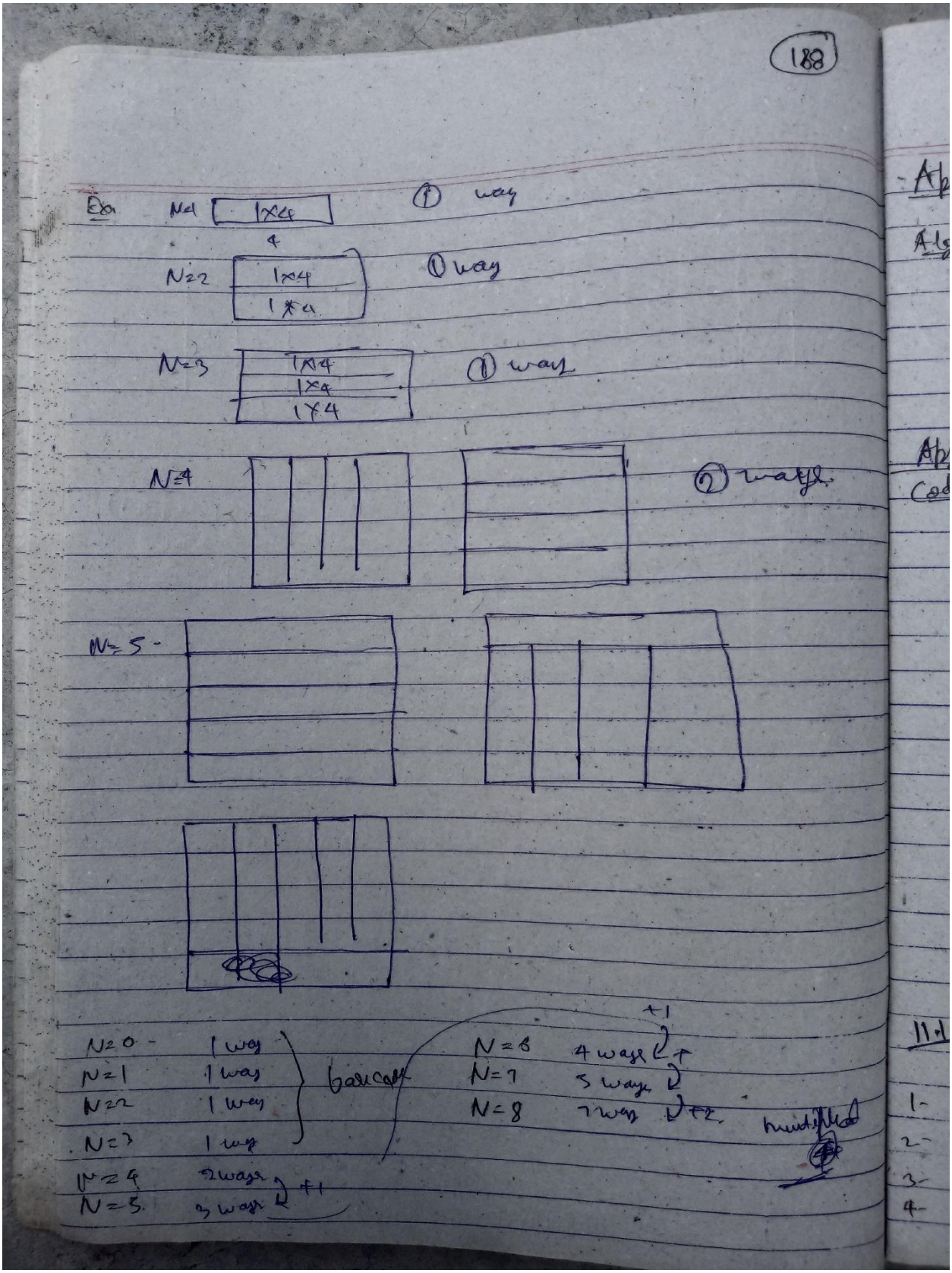
Code -

```
int minCost(int cost[R][C], int m, int n)
{
    int i, j;
    int tc[R][C];
    tc[0][0] = cost[0][0]
    for(i=1; i<=m; i++)
        tc[i][0] = tc[i-1][0] + cost[i][0];
    for(j=1; j<=n; j++)
        tc[0][j] = tc[0][j-1] + cost[0][j];
    for(i=1; i<=m; i++)
        for(j=1; j<=n; j++)
            tc[i][j] = min(tc[i-1][j-1], tc[i-1][j], tc[i][j-1]) + cost[i][j];
    return tc[m][n];
}
```

11.9 Fill a N×4 wall with 1*4 bricks problem



Task - Number of ways that we can transit.


188
Ex
N=1
1×4
1 way
N=2
1×4
1×4
1 way
N=3
1×4
1×4
1×4
1 way
N=4
2 ways
N=5
N=0 - 1 way
N=1 1 way
N=2 1 way
N=3 1 way
N=4 2 ways
N=5 3 ways
+1
base case
N=6 4 ways
N=7 5 ways
N=8 7 ways
+2

(89)

Approach 1 - Use Recursion

Algo NumWaysPanel(n).

```
if n=0 or 1 or 2 or 3
    return 1
else
    return NumberWaysPanel(n-1) + NumWaysPanel(n-4)
```

Approach 2 - Use D.P: T[n] = O(n).

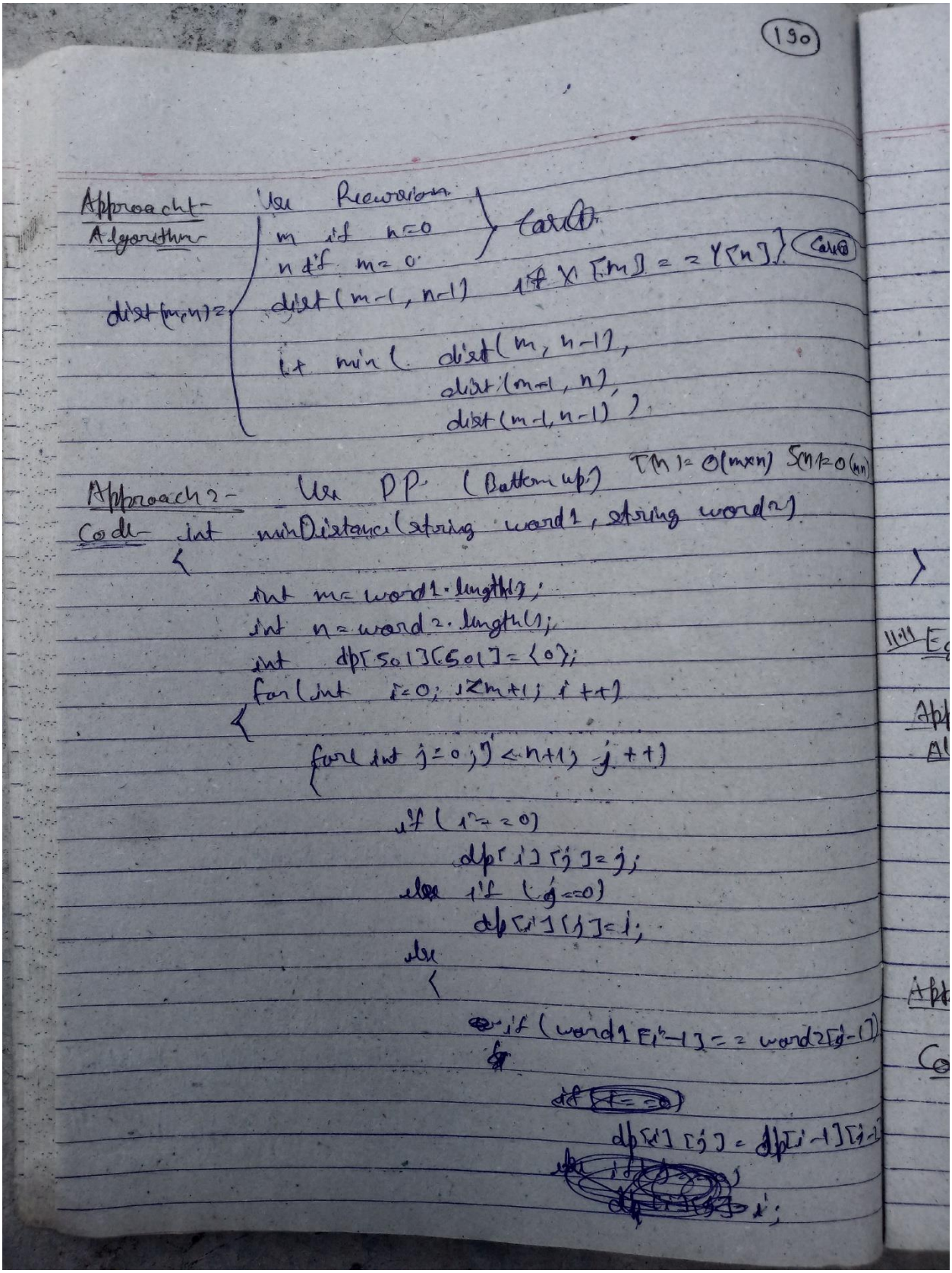
Code -

```
int count(int n)
{
    int dp[n+1];
    dp[0] = 0;
    for(int i=1; i<=n; i++)
    {
        if(i>=1 && i<=3)
            dp[i] = 1;
        else if(i==4)
            dp[i] = 2;
        else
            dp[i] = dp[i-1] + dp[i-4];
    }
    return dp[n];
}
```

11.10 Levenstein / Edit Distance Problem -

	Xm	Yn	Task - Min No. of operation to obtain Y from X
1-	Kitten	Sitten	replace K with S.
2-	Sitten	Sittin	replace e with i
3-	Sittin	Sitting	insert g
4-	Sitting	Sittin	delete g.

} 3 operation


190
Approach1- Use Recursion
Algorithm-
m if n=0
n if m=0
dist(m-1, n-1) if X[m] == Y[n]
dist(m,n) =
1+ min(dist(m, n-1),
dist(m-1, n),
dist(m-1, n-1));
Approach2- Use DP (Bottom up) T(n) = O(m×n) S(n) = O(mn)
Code- int minDistance(string word1, string word2)
{
int m = word1.length();
int n = word2.length();
int dp[501][501] = {0};
for(int i=0; i<m+1; i++)
{
for(int j=0; j<n+1; j++)
if(i==0)
dp[i][j] = j;
else if(j==0)
dp[i][j] = i;
else
{
if(word1[i-1] == word2[j-1])
dp[i][j] = dp[i-1][j-1]

```
else
{
    int add=dp[i][j-1];
    int replace=dp[i-1][j-1];
    int remove=dp[i-1][j];
    dp[i][j]=1+min(min(add,replace),
                   remove);
}
}
}
}
return dp[m][n];
}
```

Egg Droping Problem- Find breakable floor using min. no. of trails.

Approach 1- Use Recursion

Algo

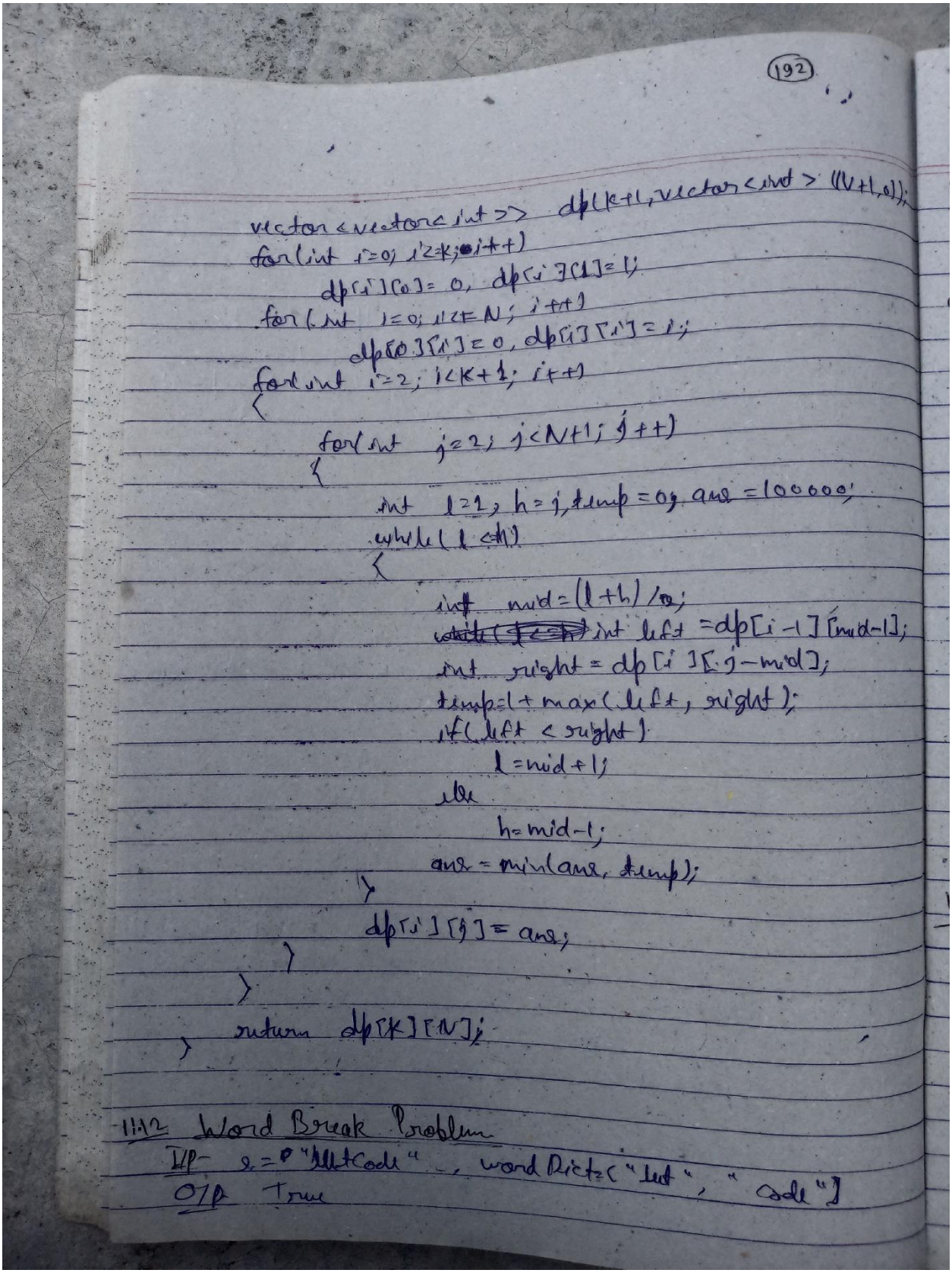
$$d(n,k) = \begin{cases} 0 & \text{if } k \le 0 \\ 1 & \text{if } k = 1 \\ k & \text{if } n = 1 \\ \text{Not solvable} & \text{if } n < 0 \\ 1 + \min(\max(d(n-1, x-1), d(n, k-x))) & \end{cases}$$

n: eggs
k: floors

Approach 2- Use DP with Binary Search.

$T(n) = O(nk \lg n)$ $S(n) = O(n \times k)$

Code-
```
int superEggDrop(int K, int N)
{
    if(N==0 || N==1)
        return N;
    if(K==1)
        return N;
```

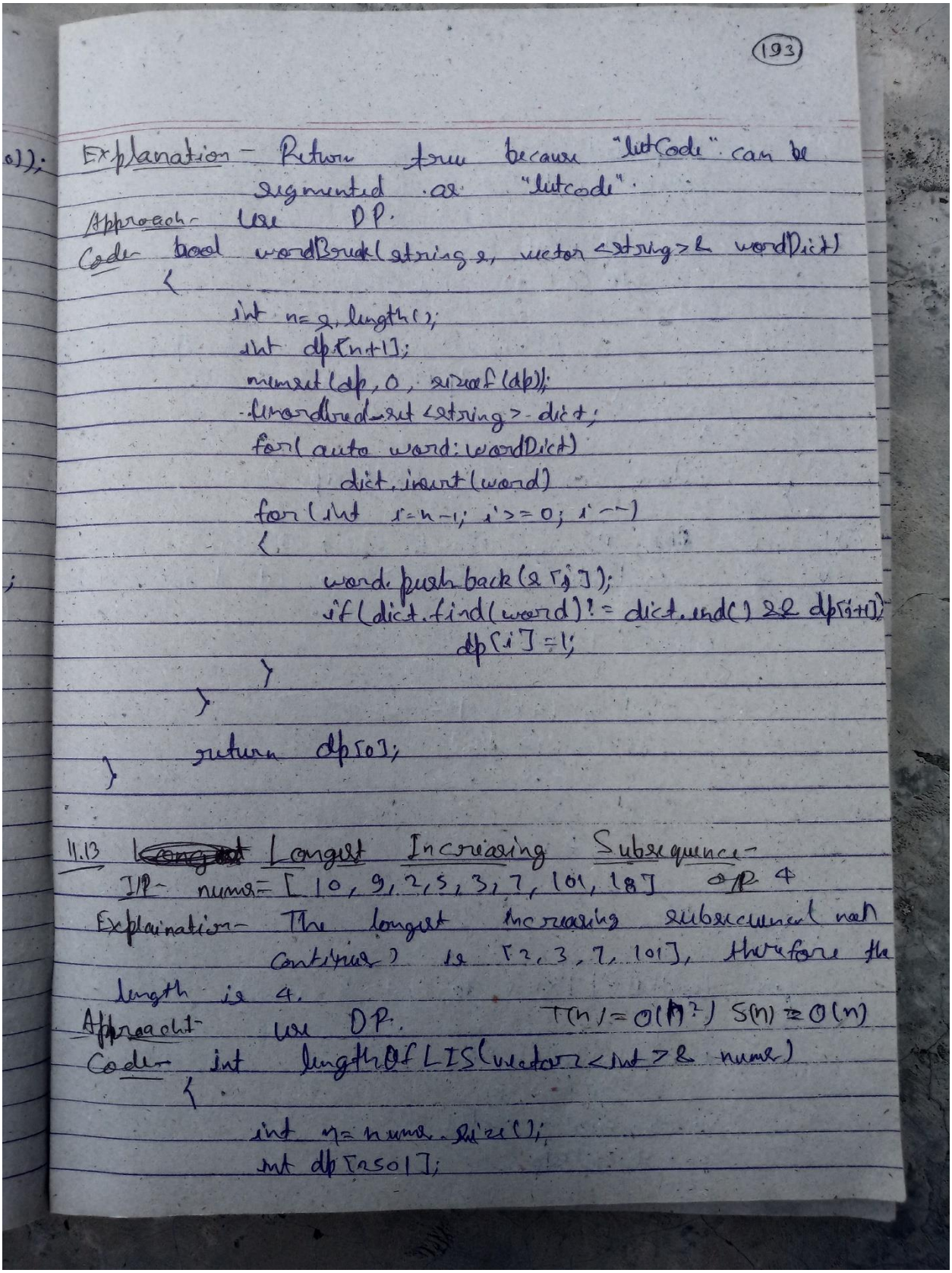


```
vector<vector<int>> dp(k+1, vector<int>(n+1,0));
for(int i=0; i<=k; i++)
    dp[i][0]=0, dp[i][1]=1;
for(int i=0; i<=N; i++)
    dp[0][i]=0, dp[1][i]=i;
for(int i=2; i<k+1; i++)
{
    for(int j=2; j<N+1; j++)
    {
        int l=1, h=j, temp=0, ans=100000;
        while(l<=h)
        {
            int mid=(l+h)/2;
            int left=dp[i-1][mid-1];
            int right=dp[i][j-mid];
            temp=1+max(left, right);
            if(left<right)
                l=mid+1;
            else
                h=mid-1;
            ans=min(ans, temp);
        }
        dp[i][j]=ans;
    }
}
return dp[k][N];
}
```

11.12 Word Break Problem

I/P- s="leetcode", wordDict=["leet", "code"]

O/P True



193

Explanation - Return true because "leetcode" can be segmented as "leet code".

Approach - Use DP.

Code -

```
bool wordBreak(string s, vector<string>& wordDict)
{
    int n = s.length();
    int dp[n+1];
    memset(dp, 0, sizeof(dp));
    unordered_set<string> dict;
    for(auto word: wordDict)
        dict.insert(word)
    for(int i=n-1; i>=0; i--)
    {
        word.push_back(s[i]);
        if(dict.find(word)!=dict.end() && dp[i+1])
            dp[i]=1;
    }
}
return dp[0];
}
```

11.13 ~~Longest~~ Longest Increasing Subsequence -

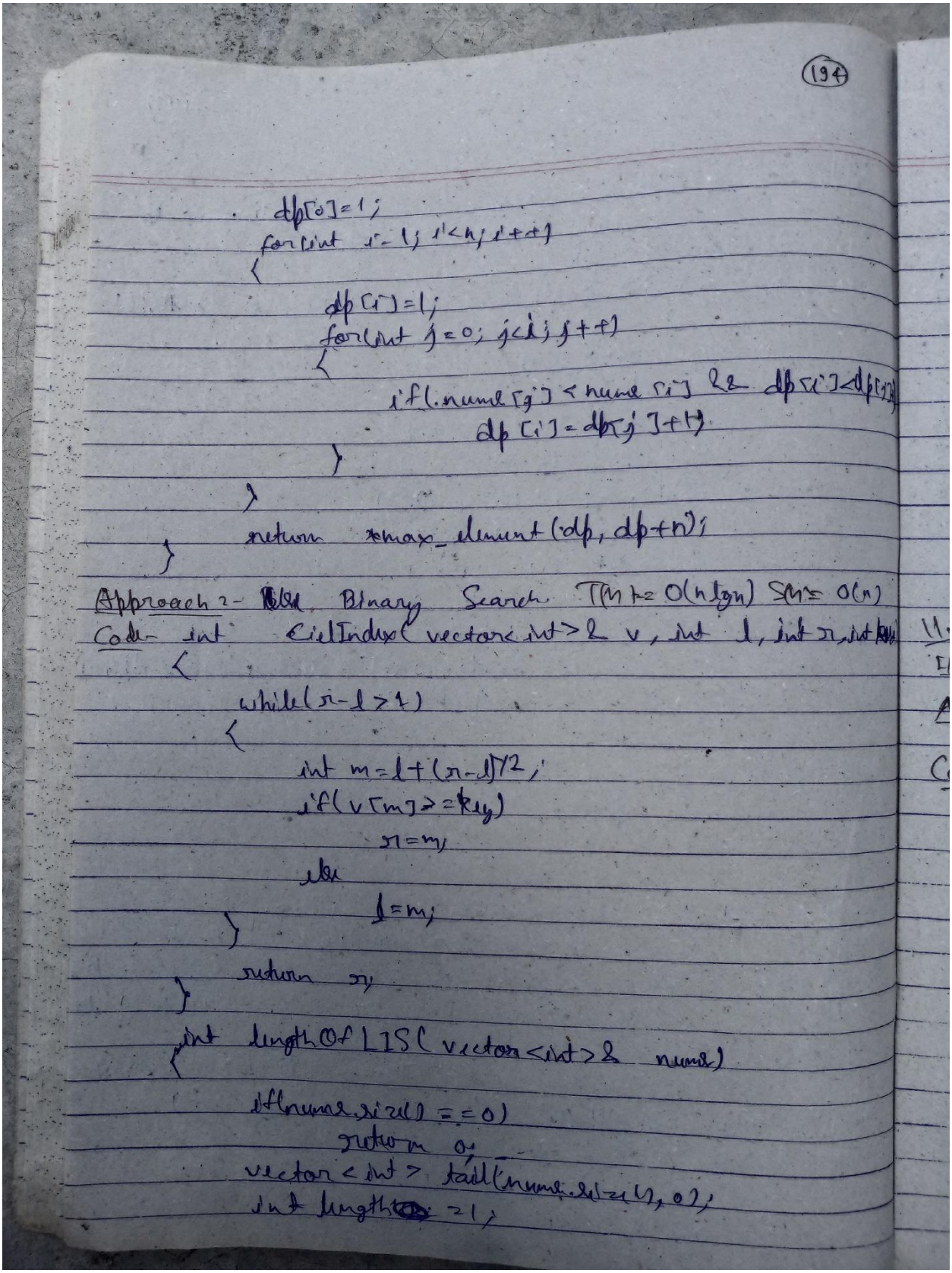
I/P - nums = [10, 9, 2, 5, 3, 7, 101, 18] O/P 4

Explanation - The longest increasing subsequence (not continuous) is [2, 3, 7, 101], therefore the length is 4.

Approach - Use DP. $T(n) = O(n^2)$ $S(n) = O(n)$

Code -

```
int lengthOfLIS(vector<int>& nums)
{
    int n = nums.size();
    int dp[n+1];
```



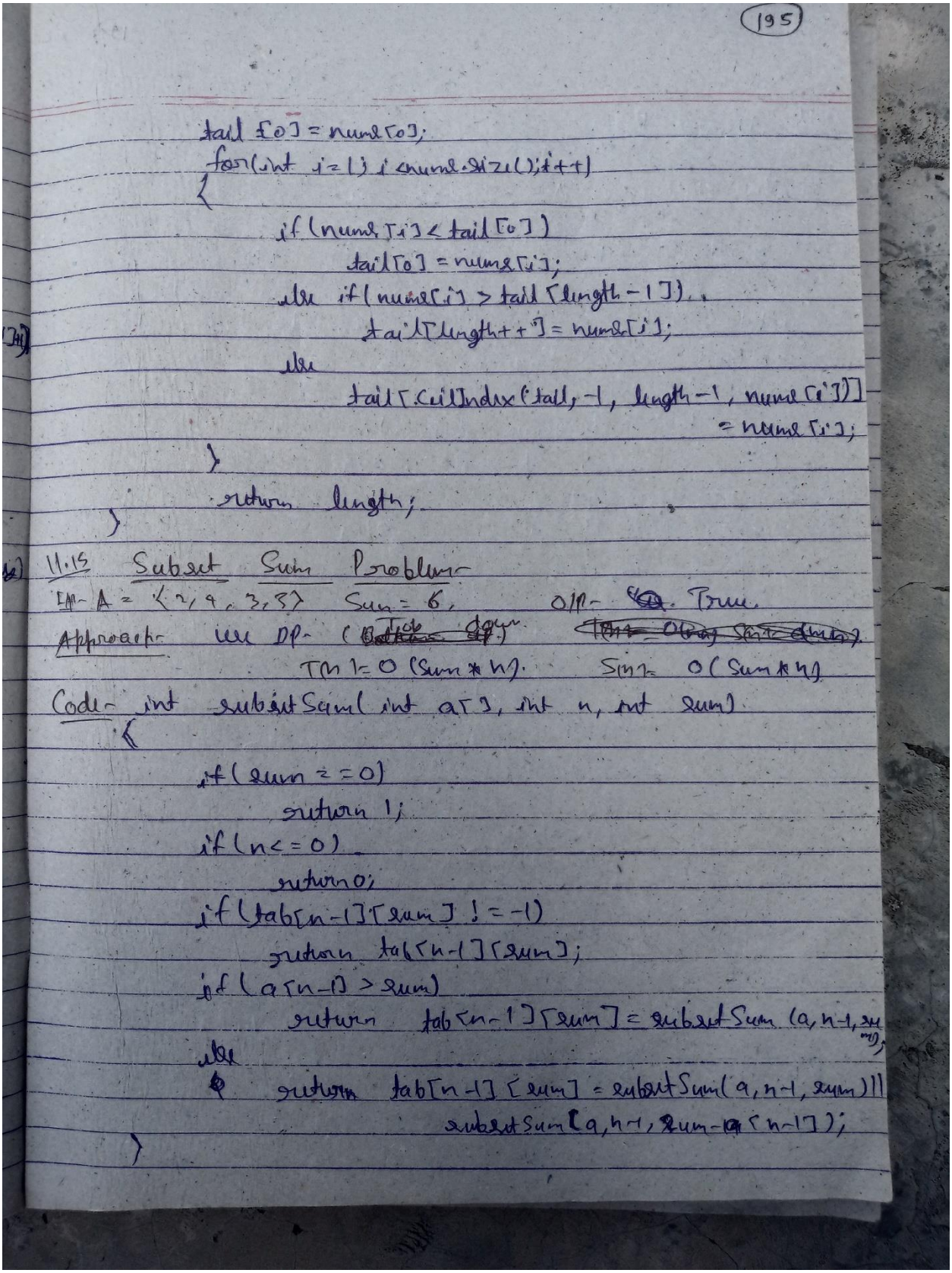
(194)

```
dp[0]=1;
for(int i=1; i<n; i++)
{
    dp[i]=1;
    for(int j=0; j<i; j++)
    {
        if(nums[j]<nums[i] && dp[i]<dp[j]+1)
            dp[i]=dp[j]+1;
    }
}
return *max_element(dp, dp+n);
}
```

Approach 2- Binary Search. T(n)= O(n log n) S(n)= O(n)

Code-

```
int CeilIndex(vector<int>& v, int l, int r, int key)
{
    while(r-l>1)
    {
        int m=l+(r-l)/2;
        if(v[m]>=key)
            r=m;
        else
            l=m;
    }
    return r;
}
int lengthOfLIS(vector<int>& nums)
{
    if(nums.size()==0)
        return 0;
    vector<int> tail(nums.size(),0);
    int length=1;
```



195

```
tail[0] = nums[0];
for(int i=1; i<nums.size(); i++)
{
    if(nums[i] < tail[0])
        tail[0] = nums[i];
    else if(nums[i] > tail[length-1])
        tail[length++] = nums[i];
    else
        tail[CeilIndex(tail, -1, length-1, nums[i])]
            = nums[i];
}
return length;
}
```

11.15 Subset Sum Problem-

Ex- A = {2, 4, 3, 5} Sum = 6, O/P- True.

Approach- use DP- (Top down)

TM:= O(Sum * n). SM:= O(Sum * n)

Code- int subsetSum(int a[], int n, int sum)

```
{
    if(sum == 0)
        return 1;
    if(n <= 0)
        return 0;
    if(tab[n-1][sum] != -1)
        return tab[n-1][sum];
    if(a[n-1] > sum)
        return tab[n-1][sum] = subsetSum(a, n-1, sum);
    else
        return tab[n-1][sum] = subsetSum(a, n-1, sum) ||
            subsetSum(a, n-1, sum-a[n-1]);
}
```

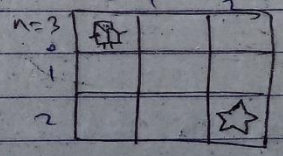
(196)

11.16 Unique Paths -

Problem Statement - A robot is located at the top-left corner of a m(Goal) x n(Goal) grid (marked start in the diagram below). The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked Finish in the diagram below).

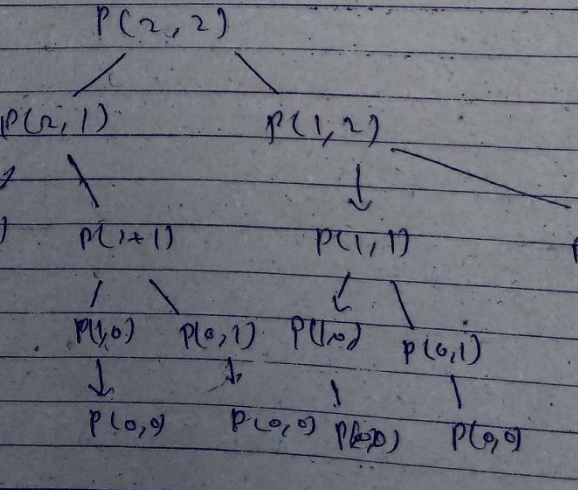
How many possible unique paths are there?

Ex



O/P = 6.

Approach :- Use recursion TM = O(2^(m+n)) SM = O(m+n)



$$P(m,n) = P(m, n-1) + P(m-1, n)$$

97

Code:
```
int uniquePaths(int m, int n)
{
    if(m==1 || n==1)
        return 1;
    return uniquePaths(m-1, n) + uniquePaths(m, n-1);
}
```

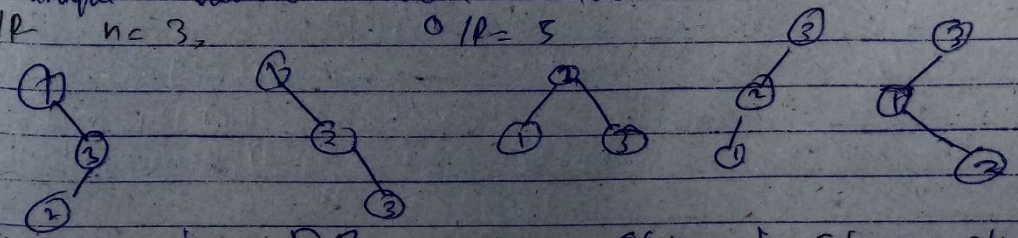
Approach:- Use DP - (Bottom up). T(n) = O(m*n) S(n) = O(m*n)

Code:
```
int uniquePaths(int m, int n)
{
    int dp[100][100];
    for(int i=0; i<m; i++)
        dp[i][0]=1;
    for(int j=0; j<n; j++)
        dp[0][j]=1;
    for(int i=1; i<m; i++)
        for(int j=1; j<n; j++)
            dp[i][j] = dp[i-1][j] + dp[i][j-1];
    return dp[m-1][n-1];
}
```

96. Unique Binary Search Trees

Problem Statement - Given an integer n, return the number of structurally unique BST's (binary search trees) which has exactly n nodes of unique values from 1 to n.

Ip- n = 3, Op = 5



Approach- Use DP. T(n) = O(n*n) S(n) = O(n)

Code:-
```
int numTrees(int n)
{
    int dp[20] = {0};
    dp[0] = 1;
    dp[1] = 1;
    for(int i=2; i<n+1; i++)
        for(int j=1; j<i+1; j++)
            dp[i] += dp[j-1] * dp[i-j];
    return dp[n];
}
```

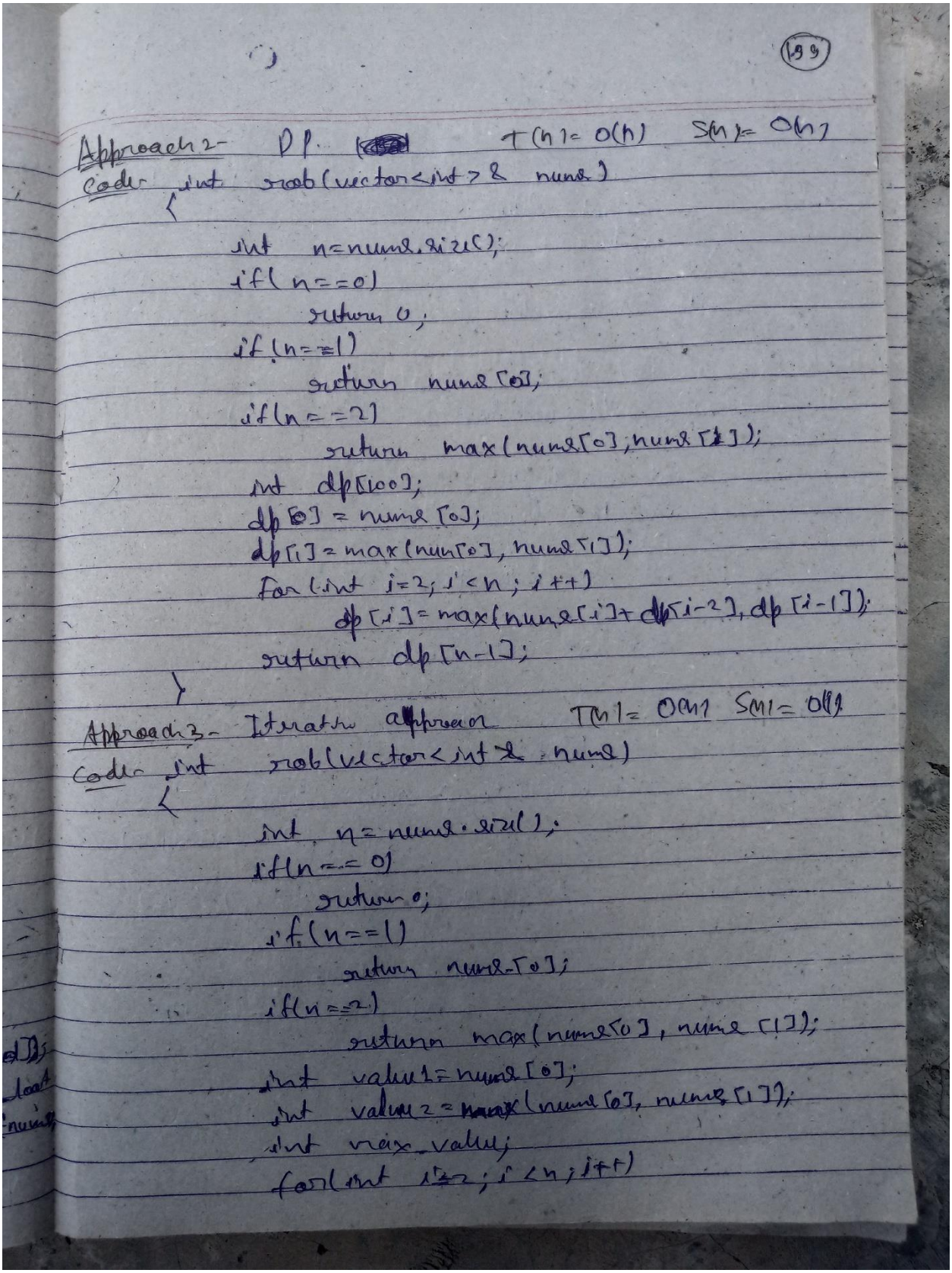
11.24 House Robber-

I/P - nums = [1, 2, 3, 1] O/P = 4.

Explanation Rob house 1 (money=1) and then rob house 3 (money = 3).
Total amount you can rob = 1+3 = 4

Approach:- Using Recursion. $T(n) = O(2^n)$.

Code:-
```
int maximize_loot(vector<int>& nums, int
                  start, int end)
{
    if(start > end)
        return 0;
    if(start == end)
        return nums[start];
    if(start+1 == end)
        return max(nums[start], nums[start+1]);
    return max(nums[start] + maximize_loot
        (nums, start+2, end), maximize_loot(nums,
        start+1, end));
}
```



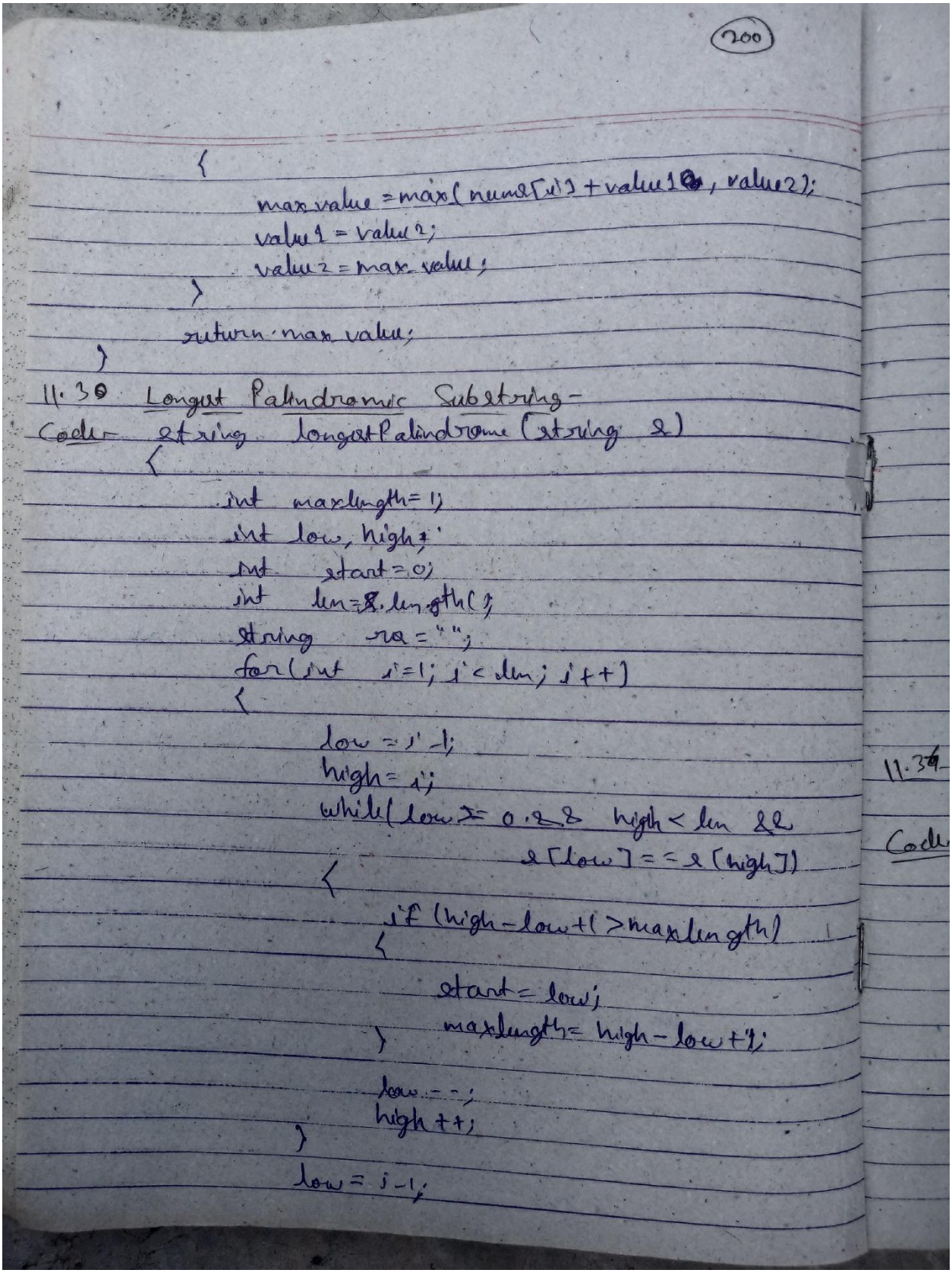
99

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

Code-
```
int rob(vector<int>& nums)
{
    int n=nums.size();
    if(n==0)
        return 0;
    if(n==1)
        return nums[0];
    if(n==2)
        return max(nums[0],nums[1]);
    int dp[100];
    dp[0] = nums[0];
    dp[1] = max(nums[0], nums[1]);
    for(int i=2; i<n; i++)
        dp[i] = max(nums[i]+dp[i-2], dp[i-1]);
    return dp[n-1];
}
```

Approach 3- Iterative approach T(n) = O(n) S(n) = O(1)

Code-
```
int rob(vector<int>& nums)
{
    int n = nums.size();
    if(n==0)
        return 0;
    if(n==1)
        return nums[0];
    if(n==2)
        return max(nums[0], nums[1]);
    int value1 = nums[0];
    int value2 = max(nums[0], nums[1]);
    int max_val;
    for(int i=2; i<n; i++)
```



200

```
    {
        max value = max( nums[i] + value1, value2);
        value1 = value2;
        value2 = max value;
    }
    return max value;
}
```

11.38 Longest Palindromic Substring -

Code -

```
string longestPalindrome (string s)
{
    int maxlength = 1;
    int low, high;
    int start = 0;
    int len = s.length();
    String ra = "";
    for(int i=1; i<len; i++)
    {
        low = i-1;
        high = i;
        while(low >= 0 && high < len &&
              s[low] == s[high])
        {
            if (high - low + 1 > maxlength)
            {
                start = low;
                maxlength = high - low + 1;
            }
            low--;
            high++;
        }
        low = i-1;
```

11.39

Code