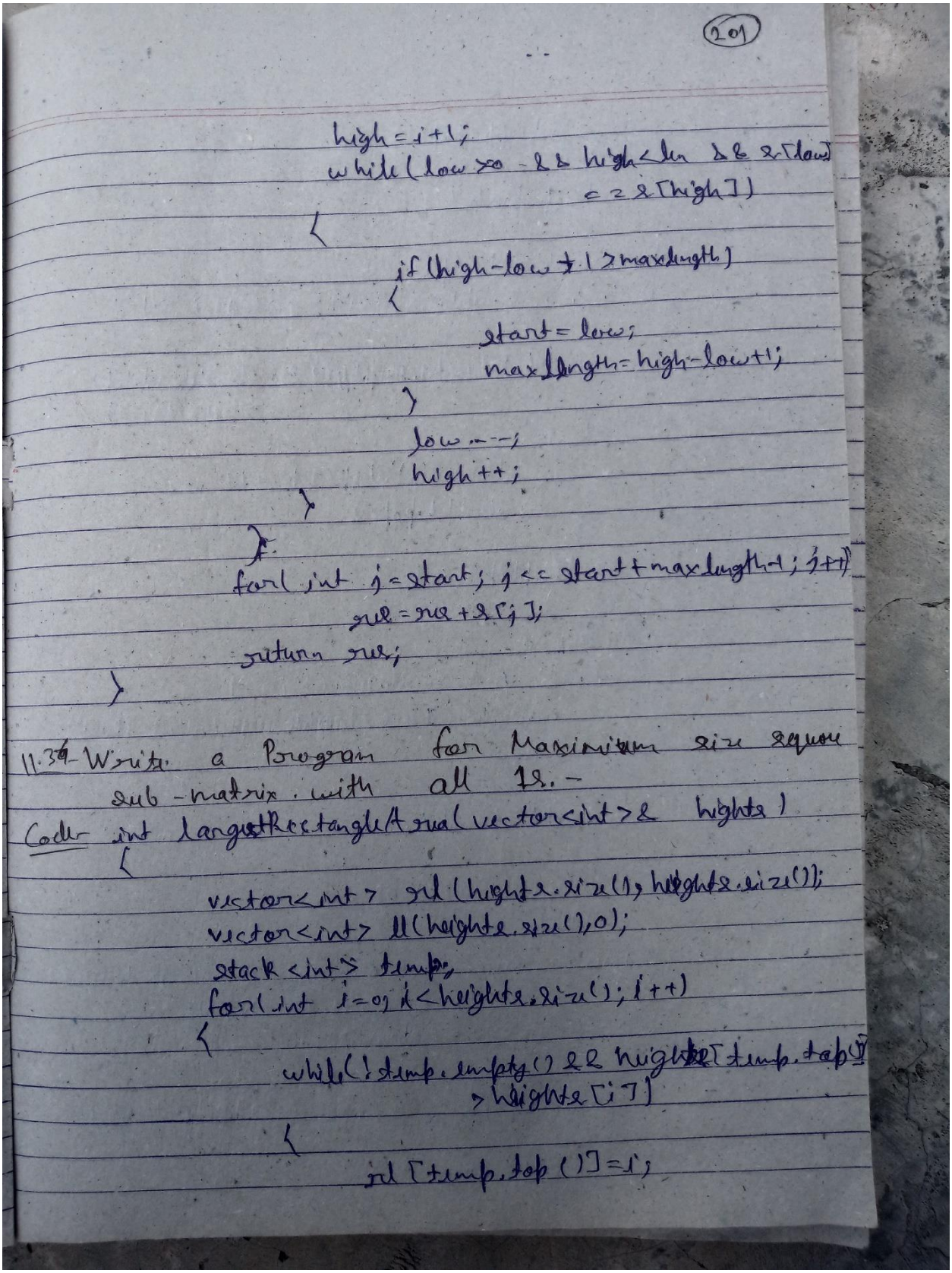
09/10/2021

Problem Solving

Applied Prep Course

SATYAM SETH
PART-3



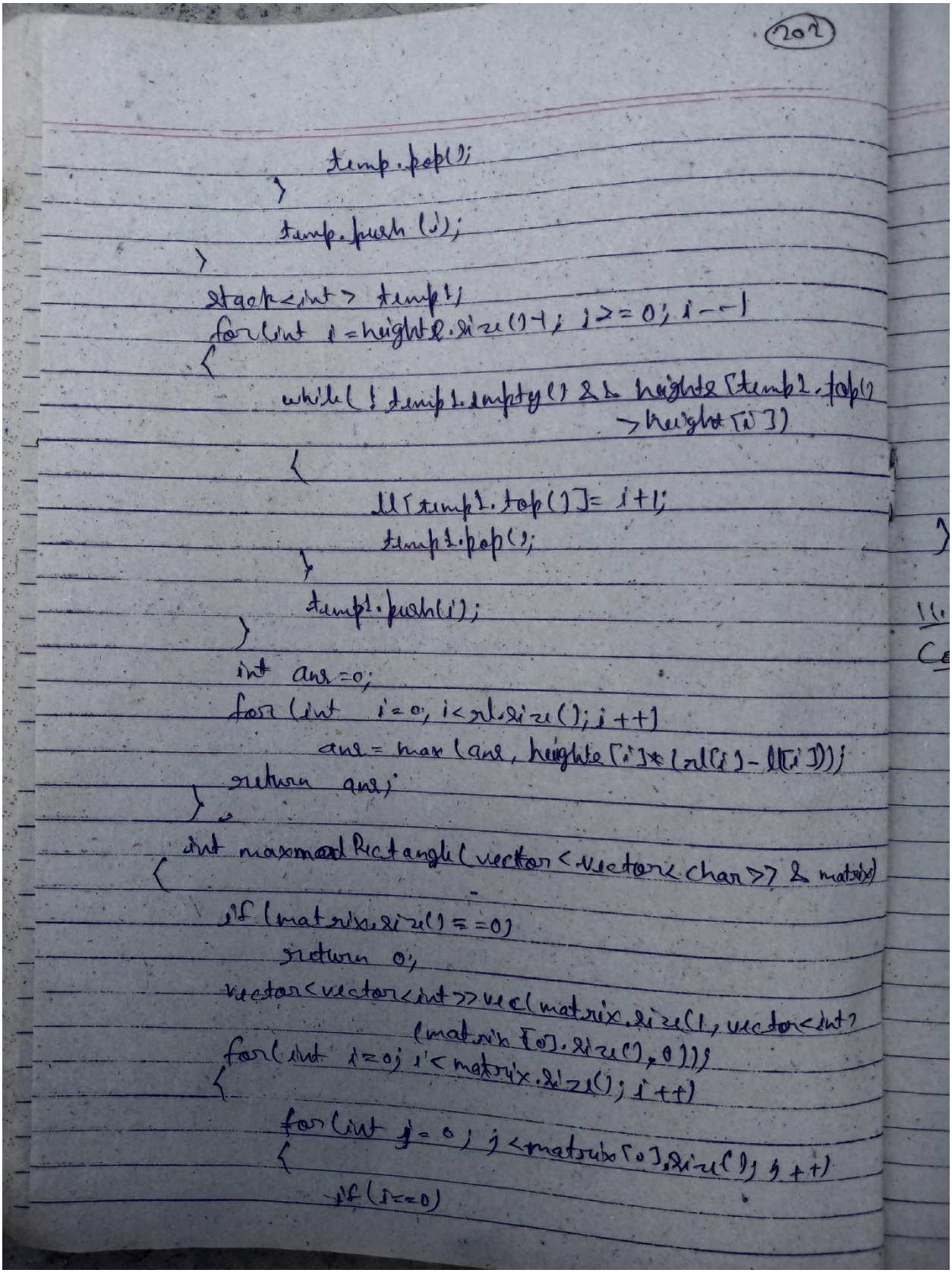
201

```
high = i+1;
while(low >=0 && high<len && arr[low]
                    == arr[high])
{
    if(high-low+1 > maxlength)
    {
        start = low;
        maxlength = high-low+1;
    }
    low--;
    high++;
}
for(int j=start; j<=start+maxlength-1; j++)
    res = res + arr[j];
return res;
}
```

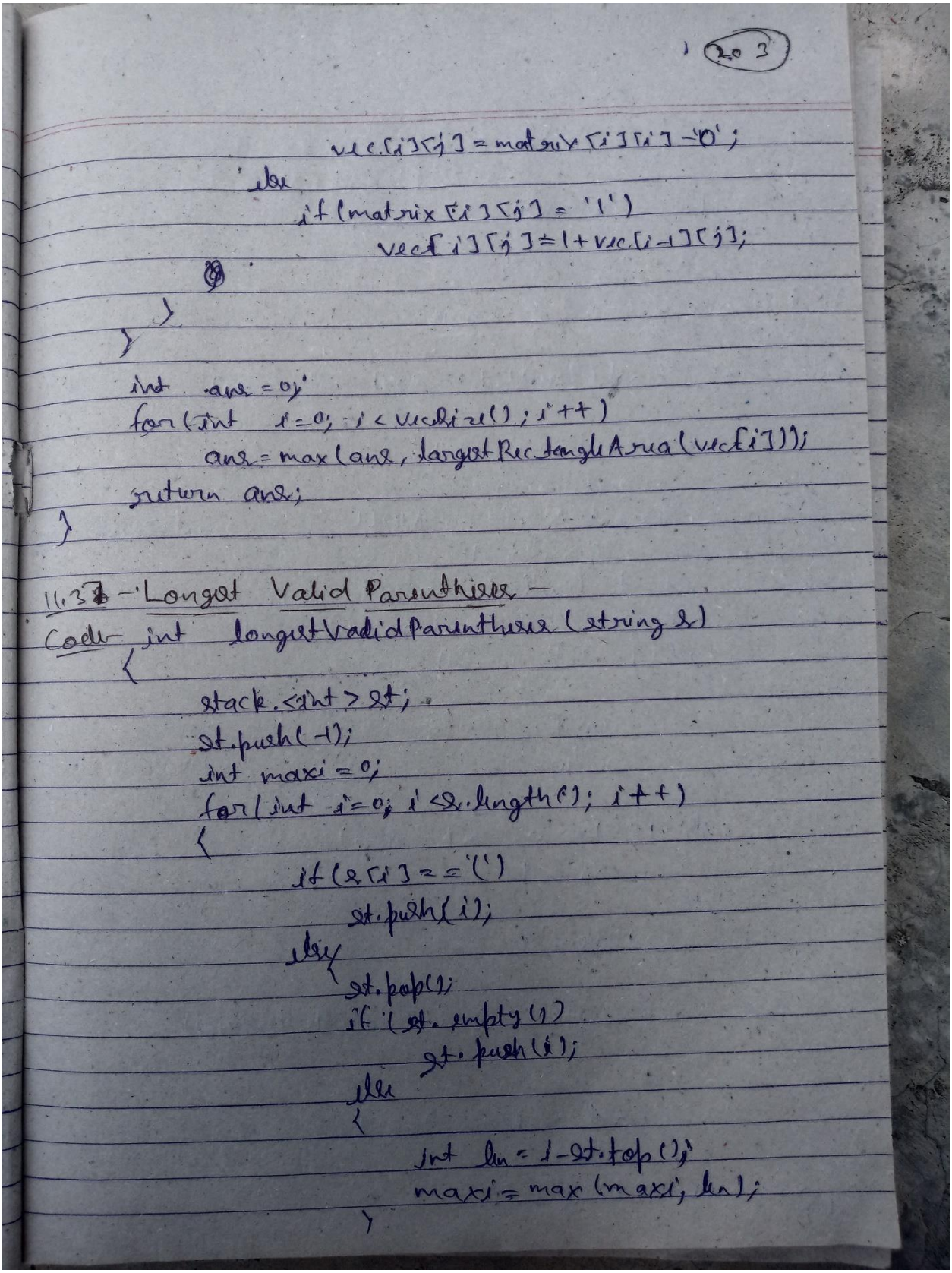
11.34 Write a Program for Maximum size square sub-matrix with all 1s.-

Code:

```
int largestRectangleArea(vector<int>& heights)
{
    vector<int> rl(heights.size(), heights.size());
    vector<int> ll(heights.size(),0);
    stack<int> temp;
    for(int i=0; i<heights.size(); i++)
    {
        while(!temp.empty() && heights[temp.top()]
                    > heights[i])
        {
            rl[temp.top()]=i;
```



```
            temp.pop();
        }
        temp.push(i);
    }
    stack<int> temp1;
    for(int i=height1.size()-1; i>=0; i--)
    {
        while(!temp1.empty() && height[temp1.top()
                                    >height[i])
        {
            ll[temp1.top()]=i+1;
            temp1.pop();
        }
        temp1.push(i);
    }
    int ans=0;
    for(int i=0; i<rl.size(); i++)
        ans=max(ans, height[i]*(rl[i]-ll[i]));
    return ans;
}
int maximalRectangle(vector<vector<char>>& matrix)
{
    if(matrix.size()==0)
        return 0;
    vector<vector<int>> vec(matrix.size(), vector<int>
                (matrix[0].size(),0));
    for(int i=0; i<matrix.size(); i++)
    {
        for(int j=0; j<matrix[0].size(); j++)
        {
            if(i==0)
```

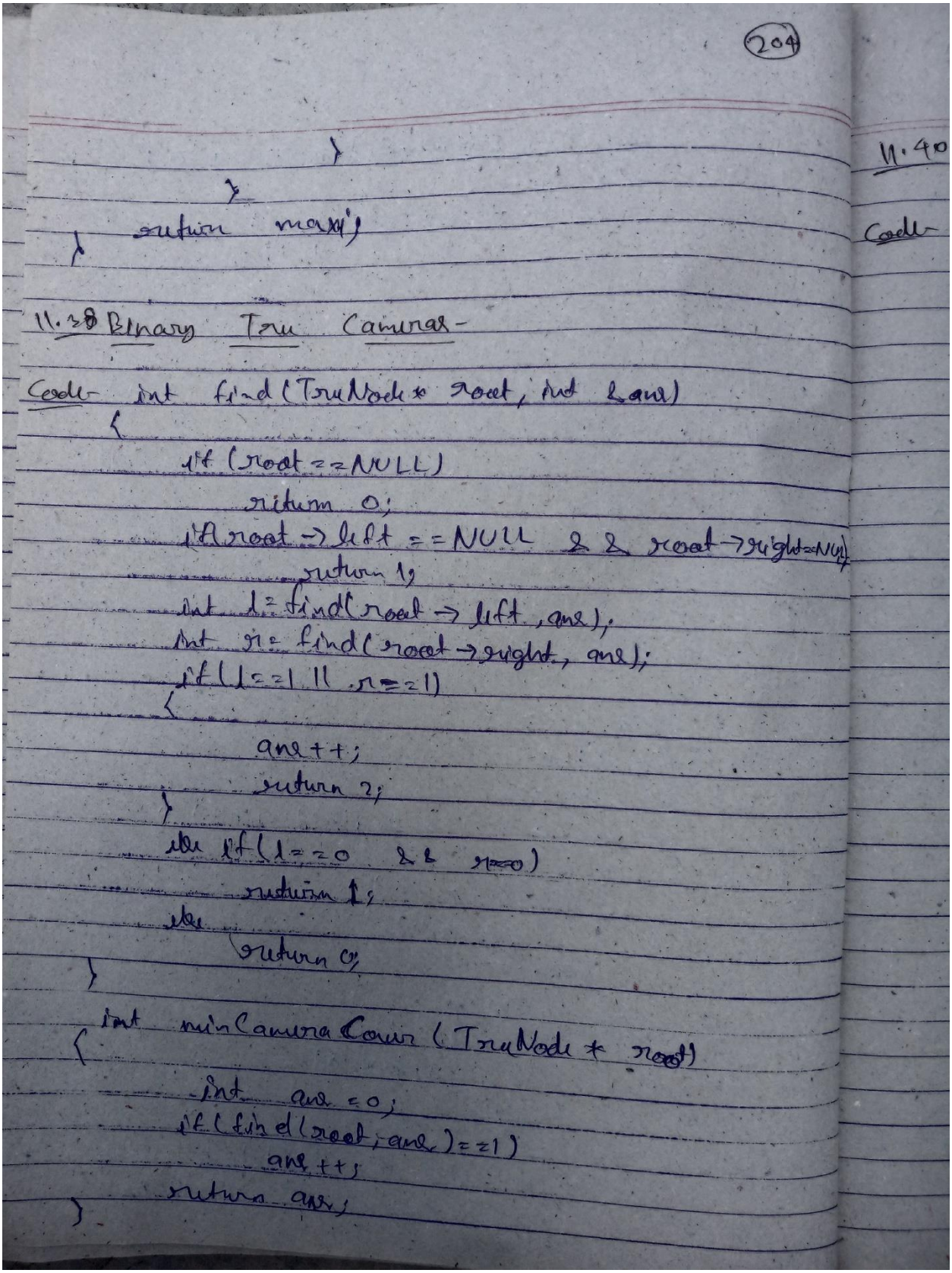


```
                vec[i][j] = matrix[i][i] - '0';
            else
                if(matrix[i][j] == '1')
                    vec[i][j] = 1 + vec[i-1][j];
        }
    }
    int ans = 0;
    for(int i=0; i<vec.size(); i++)
        ans = max(ans, largestRectangleArea(vec[i]));
    return ans;
}
```

11.32 - Longest Valid Parenthesis -

Code -

```
int longestValidParenthesis(string s)
{
    stack<int> st;
    st.push(-1);
    int maxi = 0;
    for(int i=0; i<s.length(); i++)
    {
        if(s[i] == '(')
            st.push(i);
        else
        {
            st.pop();
            if(st.empty())
                st.push(i);
            else
            {
                int len = i - st.top();
                maxi = max(maxi, len);
            }
```



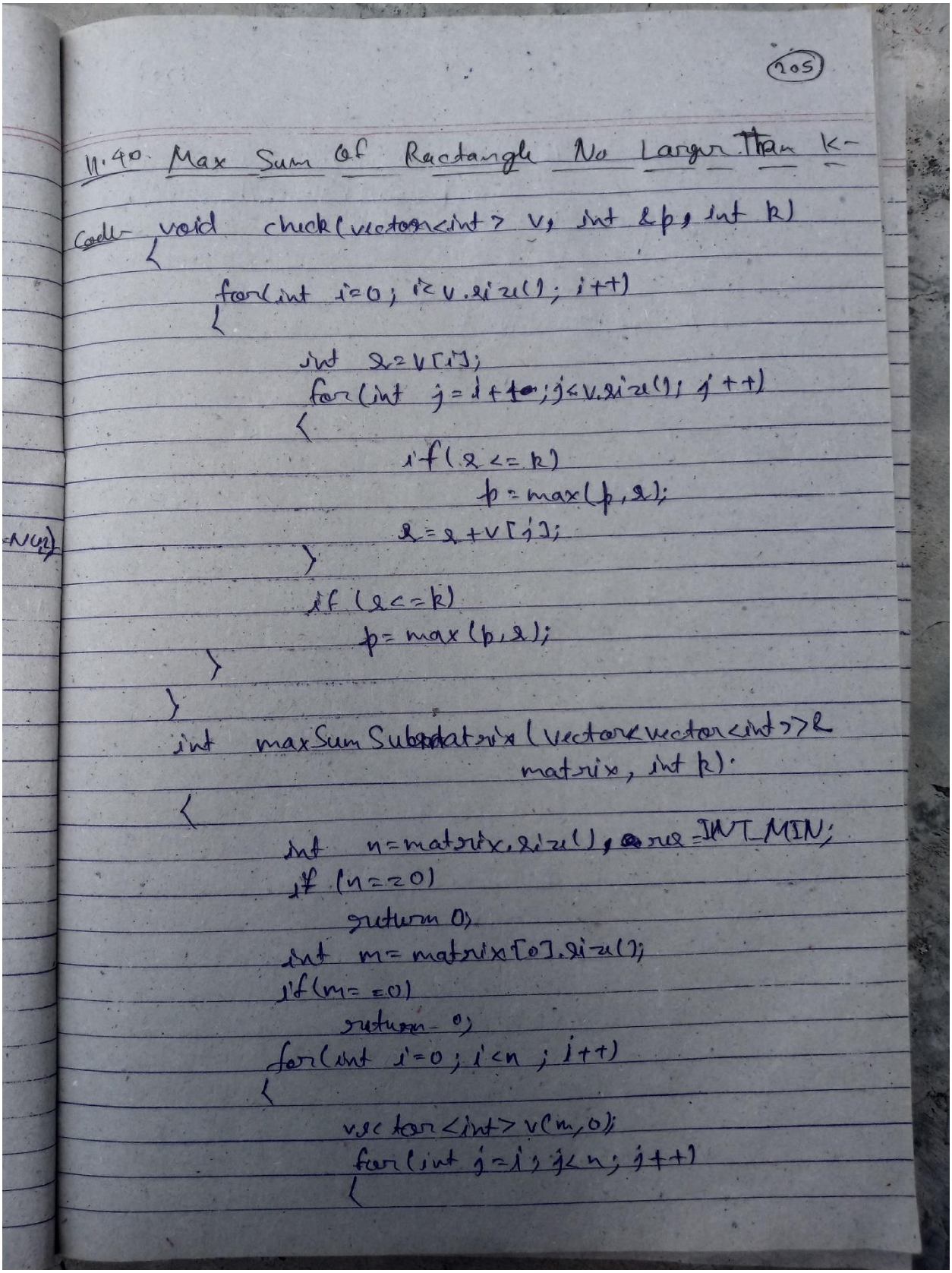
204

}
}
return max;
}

11.40
Code

11.38 Binary Tree Cameras-

Code- int find(TreeNode* root, int &ans)
{
if (root==NULL)
return 0;
if(root->left==NULL && root->right==NULL)
return 1;
int l=find(root->left, ans);
int r=find(root->right, ans);
if(l==1 || r==1)
{
ans++;
return 2;
}
else if(l==0 && r==0)
return 1;
else
return 0;
}
int minCameraCover(TreeNode* root)
{
int ans=0;
if(find(root, ans)==1)
ans++;
return ans;
}

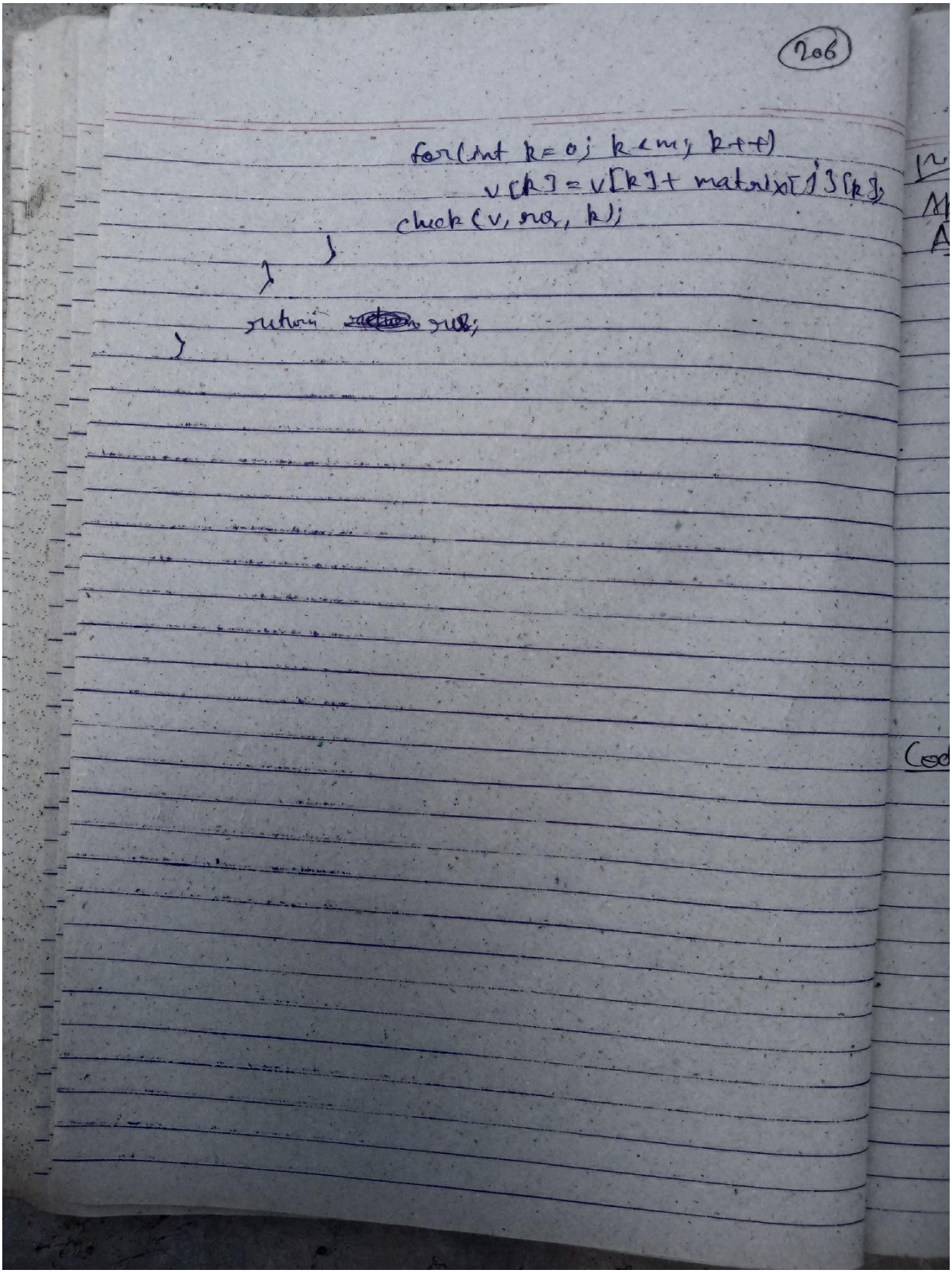


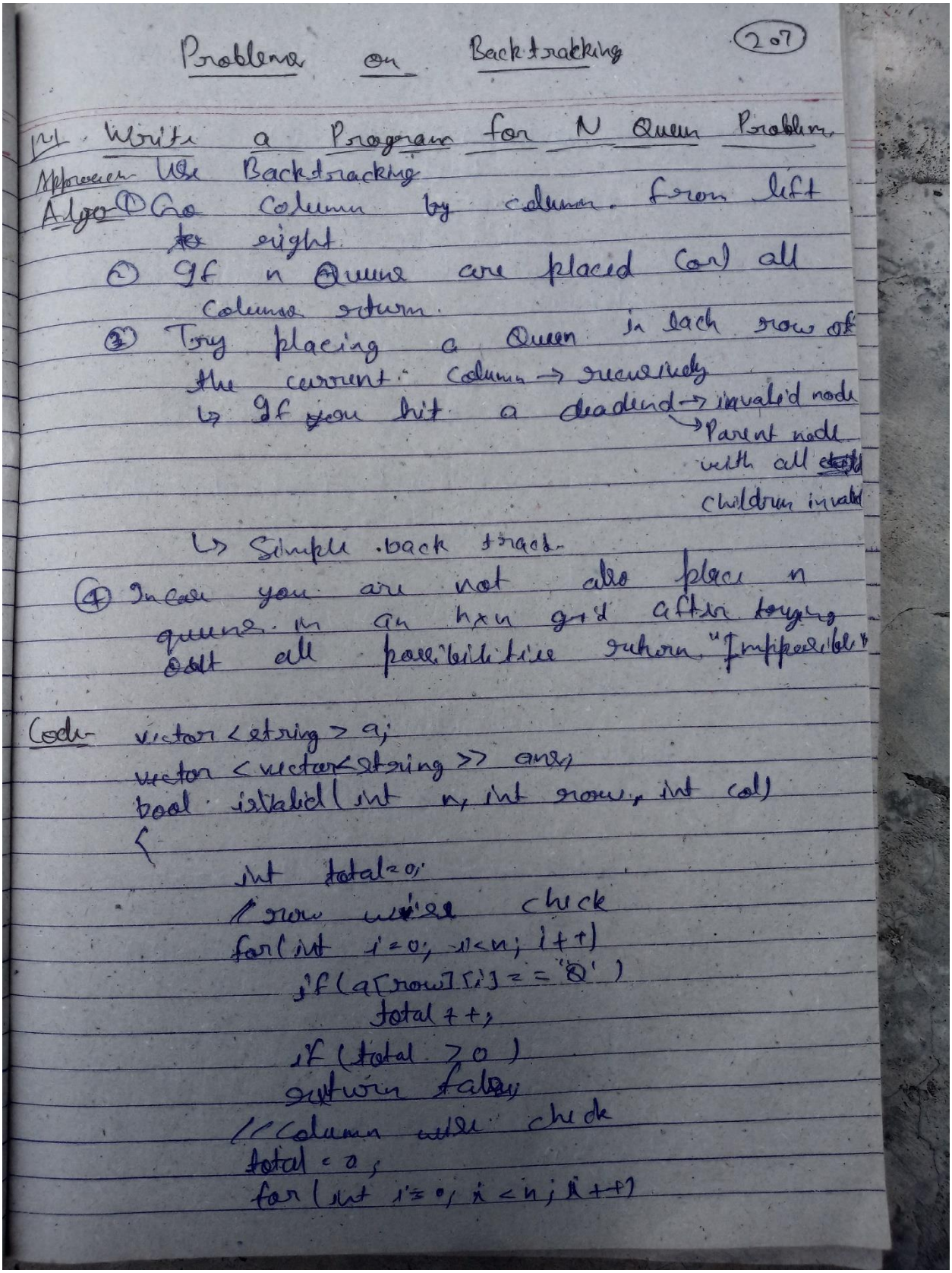
(205)

11.40 Max Sum Of Rectangle No Larger Than K-

Code-

```
void check(vector<int> v, int &p, int k)
{
    for(int i=0; i<v.size(); i++)
    {
        int s=v[i];
        for(int j=i+1; j<v.size(); j++)
        {
            if(s<=k)
                p=max(p,s);
            s=s+v[j];
        }
        if(s<=k)
            p=max(p,s);
    }
}
int maxSumSubmatrix(vector<vector<int>>&
                    matrix, int k).
{
    int n=matrix.size(), res=INT_MIN;
    if(n==0)
        return 0;
    int m=matrix[0].size();
    if(m==0)
        return 0;
    for(int i=0; i<n; i++)
    {
        vector<int> v(m,0);
        for(int j=i; j<n; j++)
        {
```

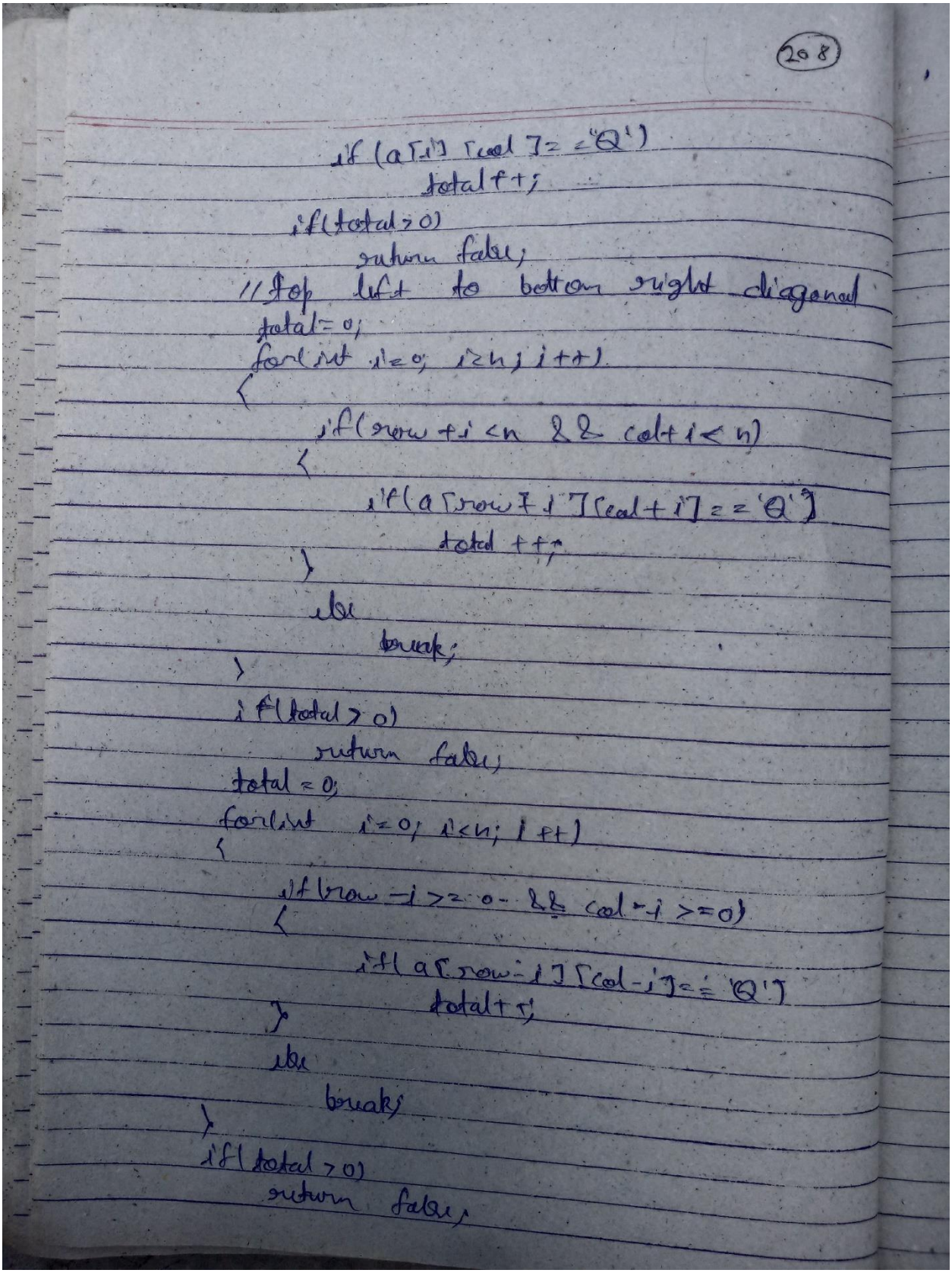

206
for(int k=0; k<m; k++)
v[k] = v[k] + matrix[i][k];
check(v, res, k);
}
}
return res;
}



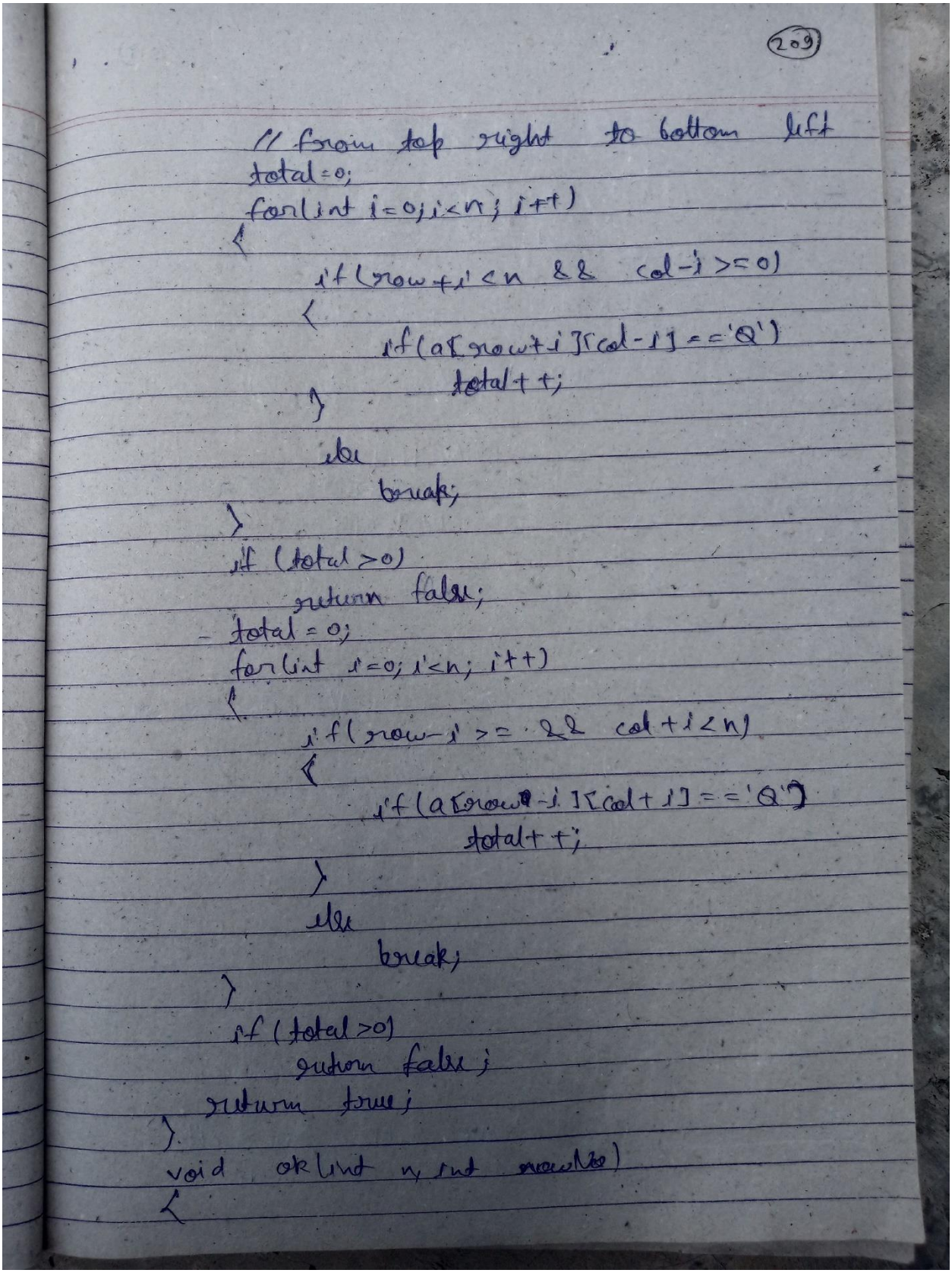
Problems on Backtracking (207)

1. Write a Program for N Queen Problem.
Approach: Use Backtracking
Algo: ① Go column by column from left to right.
② If n Queens are placed (or) all columns return.
③ Try placing a Queen in each row of the current column → recursively
↳ If you hit a deadend → invalid node → Parent node with all children invalid
↳ Simple backtrack.
④ In case you are not able place n queens in an n×n grid after trying out all possibilities return "Impossible"

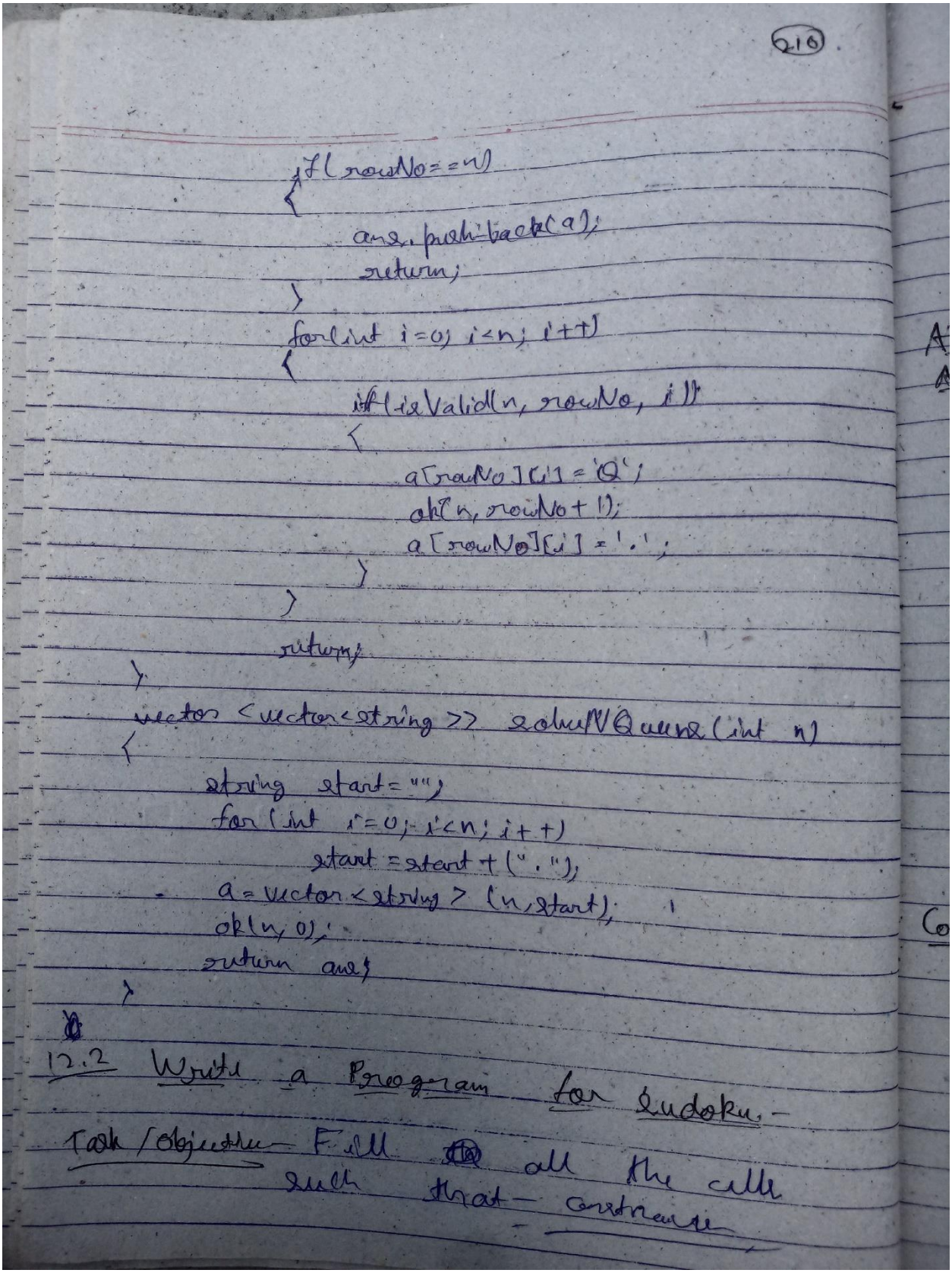
Code:
```
vector<string> a;
vector<vector<string>> ans;
bool isValid(int n, int row, int col)
{
    int total=0;
    // row wise check
    for(int i=0; i<n; i++)
        if(a[row][i]=='Q')
            total++;
    if(total>0)
        return false;
    // column wise check
    total=0;
    for(int i=0; i<n; i++)
```



```
(208)

        if (a[i][col] == 'Q')
            total++;
    if(total > 0)
        return false;
    // top left to bottom right diagonal
    total = 0;
    for(int i=0; i<n; i++)
    {
        if(row+i < n && col+i < n)
        {
            if(a[row+i][col+i] == 'Q')
                total++;
        }
        else
            break;
    }
    if(total > 0)
        return false;
    total = 0;
    for(int i=0; i<n; i++)
    {
        if(row-i >= 0 && col-i >= 0)
        {
            if(a[row-i][col-i] == 'Q')
                total++;
        }
        else
            break;
    }
    if(total > 0)
        return false;
```



```
(209)

// from top right to bottom left
total=0;
for(int i=0;i<n; i++)
{
    if(row+i<n && col-i>=0)
    {
        if(a[row+i][col-i]=='Q')
            total++;
    }
    else
        break;
}
if (total>0)
    return false;
total=0;
for(int i=0; i<n; i++)
{
    if(row-i>=0 && col+i<n)
    {
        if(a[row-i][col+i]=='Q')
            total++;
    }
    else
        break;
}
if (total>0)
    return false;
return true;
}
void ok(int n, int row)
{
```



```
(210).

if(rowNo==n)
{
    ans.push_back(a);
    return;
}
for(int i=0; i<n; i++)
{
    if(isValid(n, rowNo, i))
    {
        a[rowNo][i] = 'Q';
        ok(n, rowNo+1);
        a[rowNo][i] = '.';
    }
}
return;
}
vector<vector<string>> solveNQueens(int n)
{
    string start="";
    for(int i=0; i<n; i++)
        start = start + ".";
    a = vector<string>(n, start);
    ok(n, 0);
    return ans;
}
```

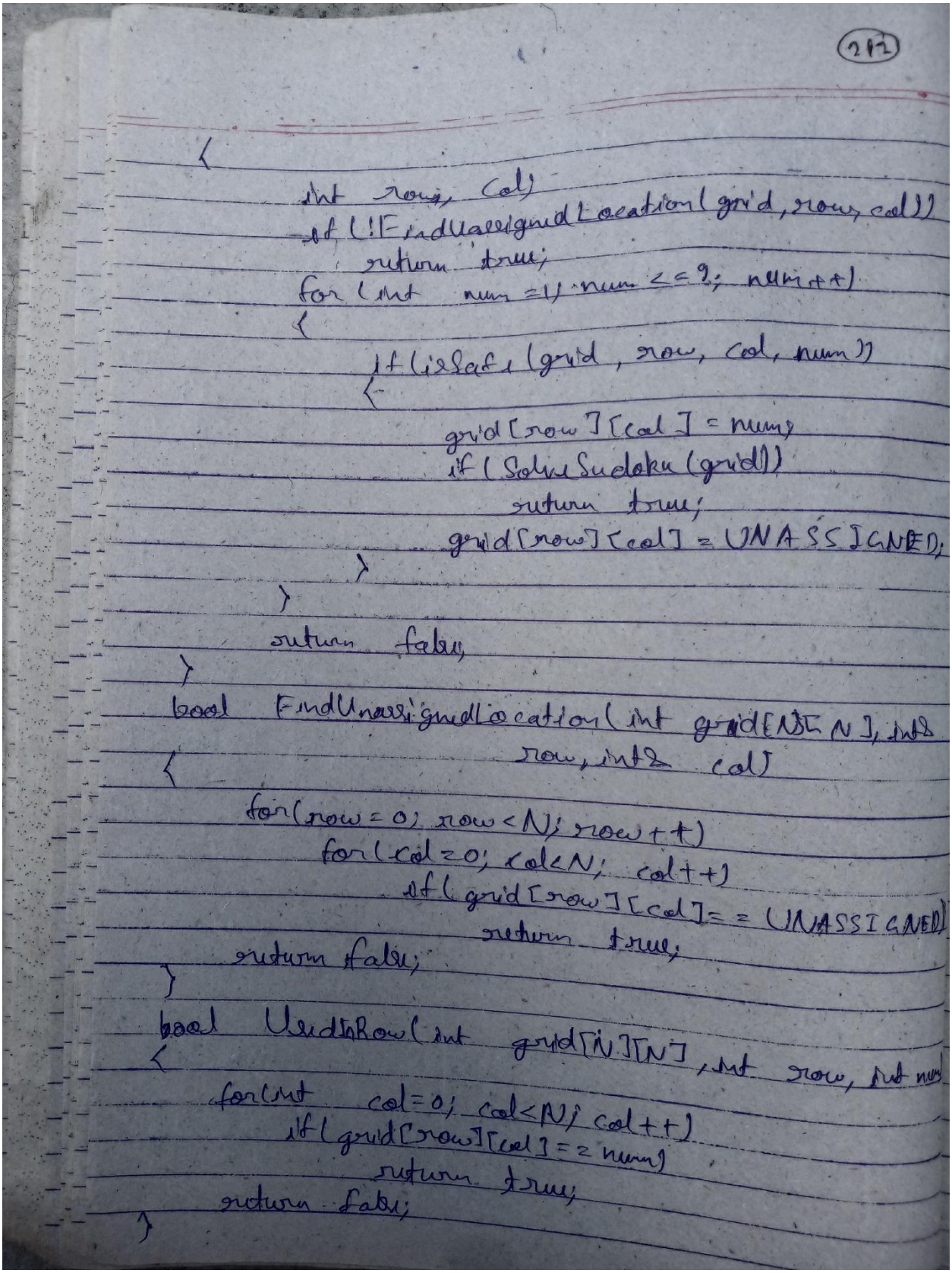
12.2 Write a Program for Sudoku.-

Task / Objective - Fill all the cells such that - constraints

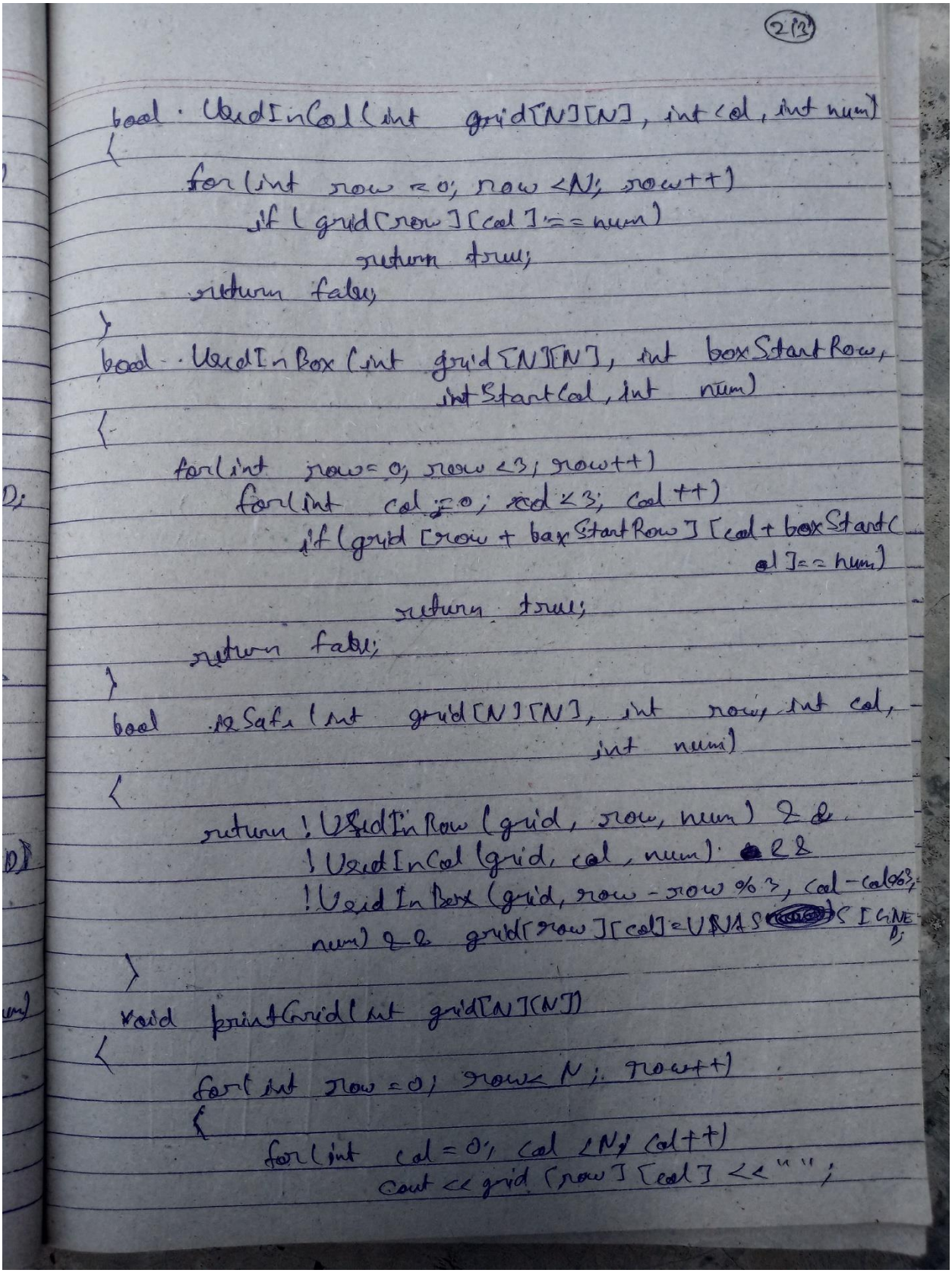
(a) Each row should contain 1 to n without repetition.
(b) Each col should contain 1 to n without repetition.
(c) Each subgrid should contain 1 to n without repetition.

Approach- Use Backtracking
Algo- (1) Make a list of all empty cells.
(2) Select an empty cell & place possible values from 1 to n one after another.
(3) Recursively expand as long as you don't hit a dead end.
↳ Node is invalid.
↳ Parent node where all children are invalid
(4) Backtrack whenever you hit a dead-end & go back to the parent node & continue
(5) Repeat 2-4 till all the cells are filled with valid values.

Code-
```
#include<bits/stdc++.h>
using namespace std;
#define UNASSIGNED 0
#define N 9
bool FindUnassignedLocation(int grid[N][N],
                            int& row, int& col);
bool isSafe(int grid[N][N], int row, int col, int
                                                num);
bool isSafe(int grid[N][N])
```



```
(21)

{
    int row, col;
    if (!FindUnassignedLocation(grid, row, col))
        return true;
    for (int num = 1; num <= 9; num++)
    {
        if (isSafe(grid, row, col, num))
        {
            grid[row][col] = num;
            if (SolveSudoku(grid))
                return true;
            grid[row][col] = UNASSIGNED;
        }
    }
    return false;
}
bool FindUnassignedLocation(int grid[N][N], int&
                            row, int& col)
{
    for(row = 0; row < N; row++)
        for(col = 0; col < N; col++)
            if(grid[row][col] == UNASSIGNED)
                return true;
    return false;
}
bool UsedInRow(int grid[N][N], int row, int num
{
    for(int col = 0; col < N; col++)
        if(grid[row][col] == num)
            return true;
    return false;
}
```

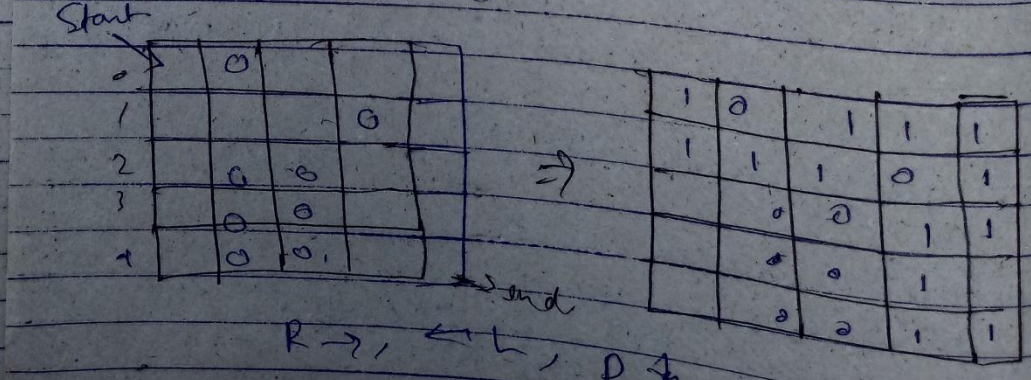


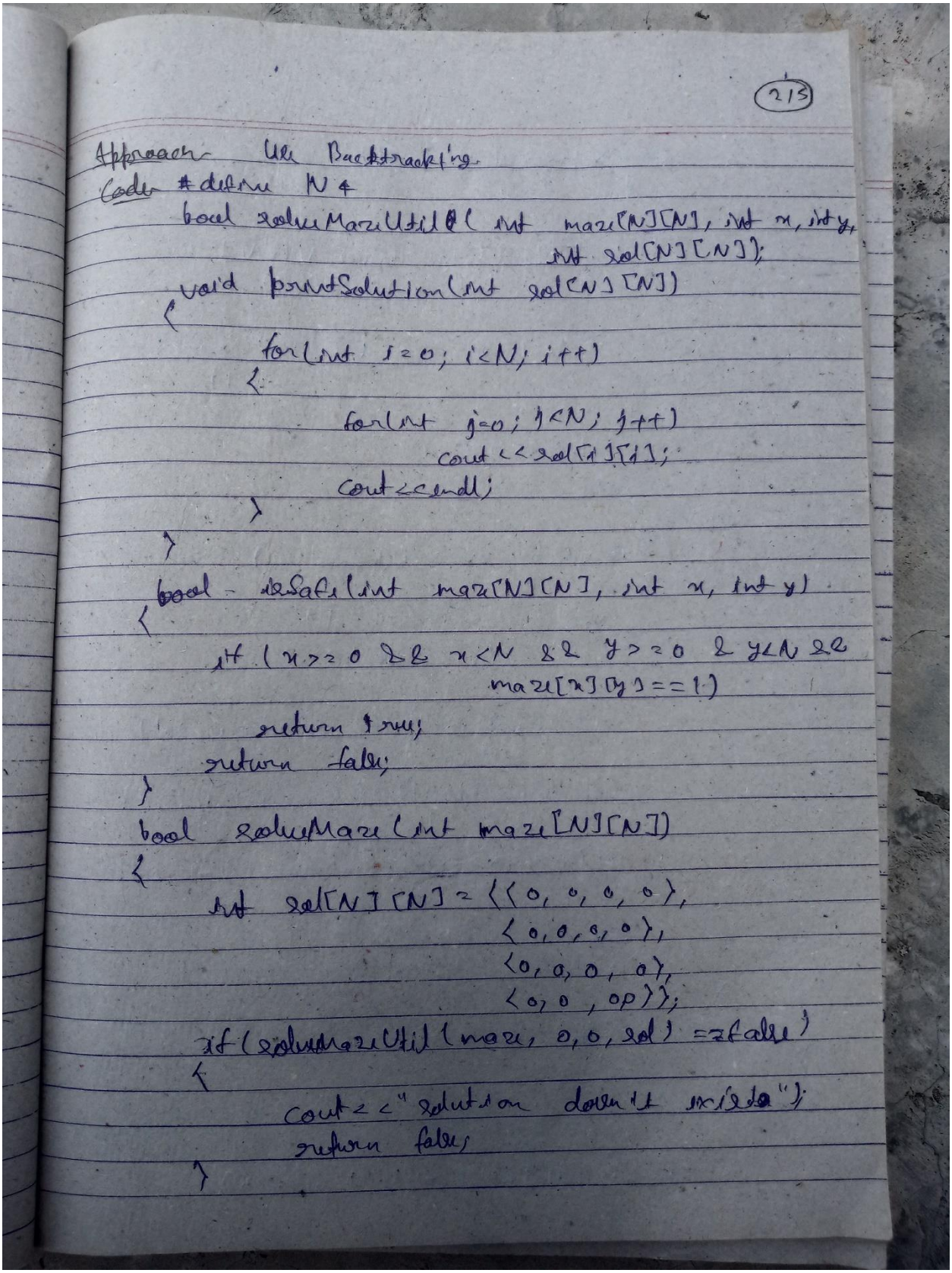
```
bool UsedInCol(int grid[N][N], int col, int num)
{
    for(int row = 0; row < N; row++)
        if(grid[row][col] == num)
            return true;
    return false;
}
bool UsedInBox(int grid[N][N], int boxStartRow,
               int StartCol, int num)
{
    for(int row = 0; row < 3; row++)
        for(int col = 0; col < 3; col++)
            if(grid[row + boxStartRow][col + boxStartCol] == num)
                return true;
    return false;
}
bool isSafe(int grid[N][N], int row, int col,
            int num)
{
    return !UsedInRow(grid, row, num) &&
           !UsedInCol(grid, col, num) &&
           !UsedInBox(grid, row - row % 3, col - col % 3,
           num) && grid[row][col] == UNASSIGNED;
}
void printGrid(int grid[N][N])
{
    for(int row = 0; row < N; row++)
    {
        for(int col = 0; col < N; col++)
            cout << grid[row][col] << " ";
```

```
        }  cout << endl;
    }
int main()
{
    int grid[N][N] = { { 3,0,6,5,0,8,4,0,0},
                       { 5,2,0,0,0,0,0,0,0},
                       { 0,8,7,0,0,0,0,3,1},
                       { 0,0,3,0,1,0,0,8,0},
                       { 9,0,0,8,6,3,0,0,5},
                       { 0,5,0,0,9,0,6,0,0},
                       { 1,3,0,0,0,0,2,5,0},
                       { 0,0,0,0,0,0,0,7,4},
                       { 0,0,5,2,0,6,3,0,0}};
    if (SolveSudoku(grid) == true)
        printGrid(grid);
    else
        cout << "No solution exists";
    return 0;
}
```

12.3 Write a Program for Rat in a Maze

Navigate a Maze.



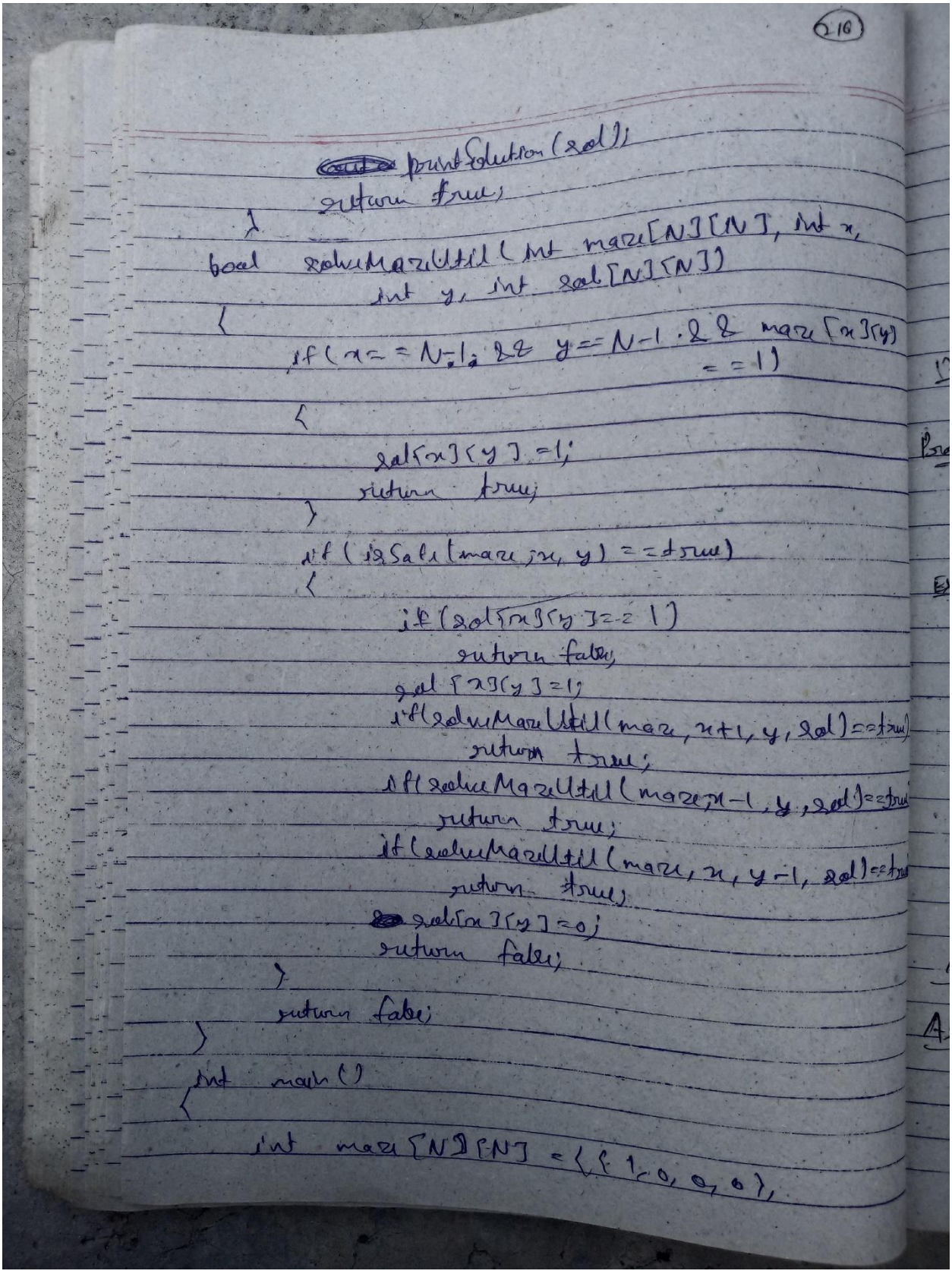


213

Approach - Use Backtracking.

Code

```
#define N 4
bool solveMazeUtil(int maze[N][N], int x, int y,
                   int sol[N][N]);
void printSolution(int sol[N][N])
{
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<N; j++)
            cout << sol[i][j];
        cout << endl;
    }
}
bool isSafe(int maze[N][N], int x, int y)
{
    if (x >= 0 && x < N && y >= 0 && y < N &&
        maze[x][y] == 1)
        return true;
    return false;
}
bool solveMaze(int maze[N][N])
{
    int sol[N][N] = {{0, 0, 0, 0},
                     {0, 0, 0, 0},
                     {0, 0, 0, 0},
                     {0, 0, 0, 0}};
    if(solveMazeUtil(maze, 0, 0, sol) == false)
    {
        cout << "Solution doesn't exists";
        return false;
    }
```



```
                                                    (210)
        printSolution(sol);
        return true;
    }
bool solveMazeUtil(int maze[N][N], int x,
        int y, int sol[N][N])
{
    if(x == N-1 && y == N-1 && maze[x][y]
                                == 1)
    {
        sol[x][y] = 1;
        return true;
    }
    if(isSafe(maze, x, y) == true)
    {
        if(sol[x][y] == 1)
            return false;
        sol[x][y] = 1;
        if(solveMazeUtil(maze, x+1, y, sol) == true)
            return true;
        if(solveMazeUtil(maze, x-1, y, sol) == true)
            return true;
        if(solveMazeUtil(maze, x, y-1, sol) == true)
            return true;
        sol[x][y] = 0;
        return false;
    }
    return false;
}
int main()
{
    int maze[N][N] = {{1, 0, 0, 0},
```

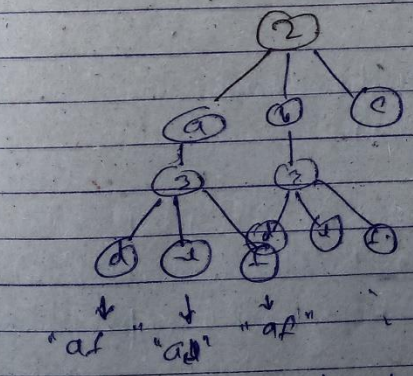
(217)

```
{1, 1, 0, 1},
{0, 1, 0, 0},
{1, 1, 1, 1}};
solveMaze(maze);
return 0;
}
```

12.4 - Letter Combinations of a phone number -

Problem Statement - Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent.

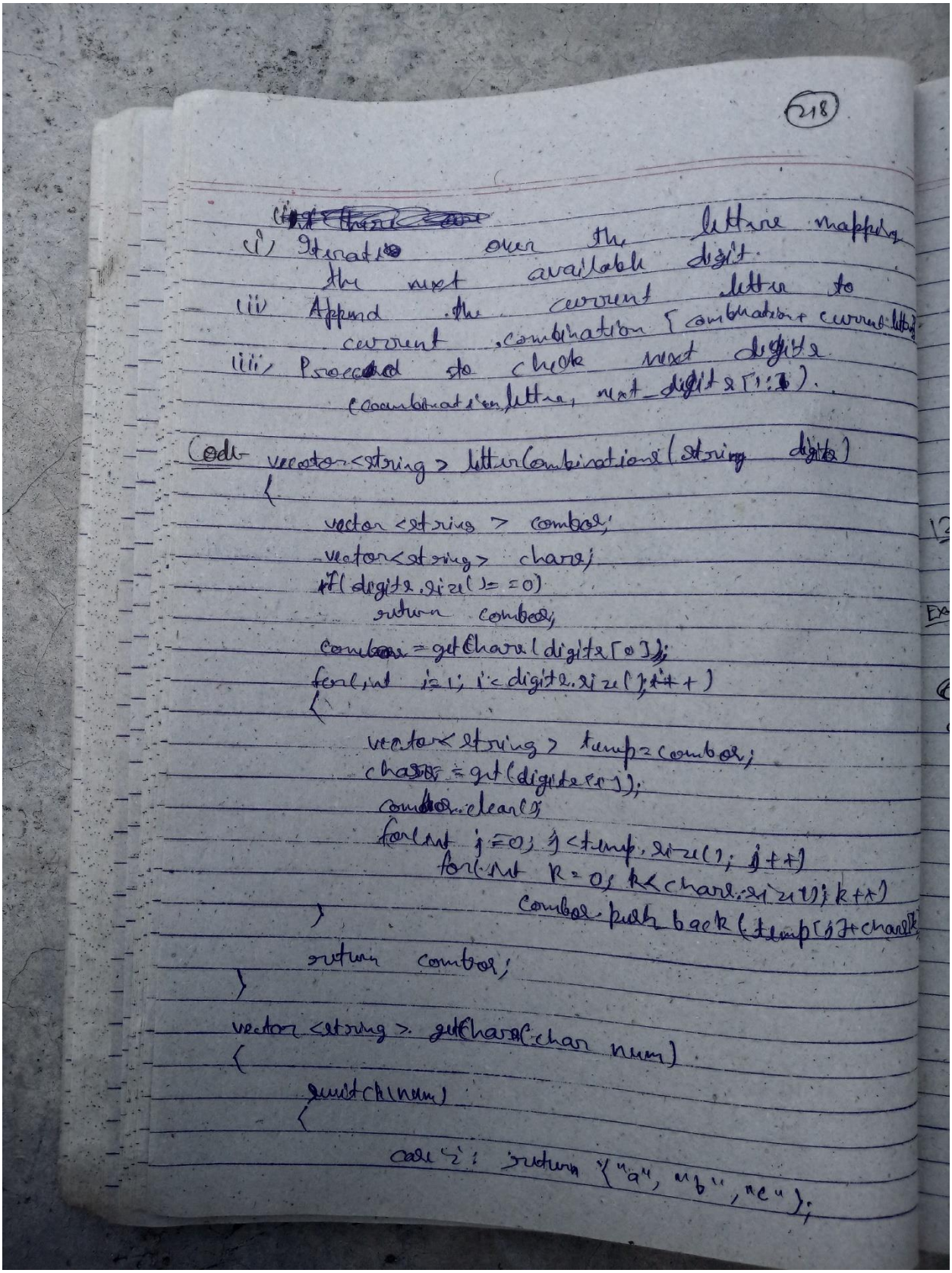
Ex. I/P - "23" O/P - ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

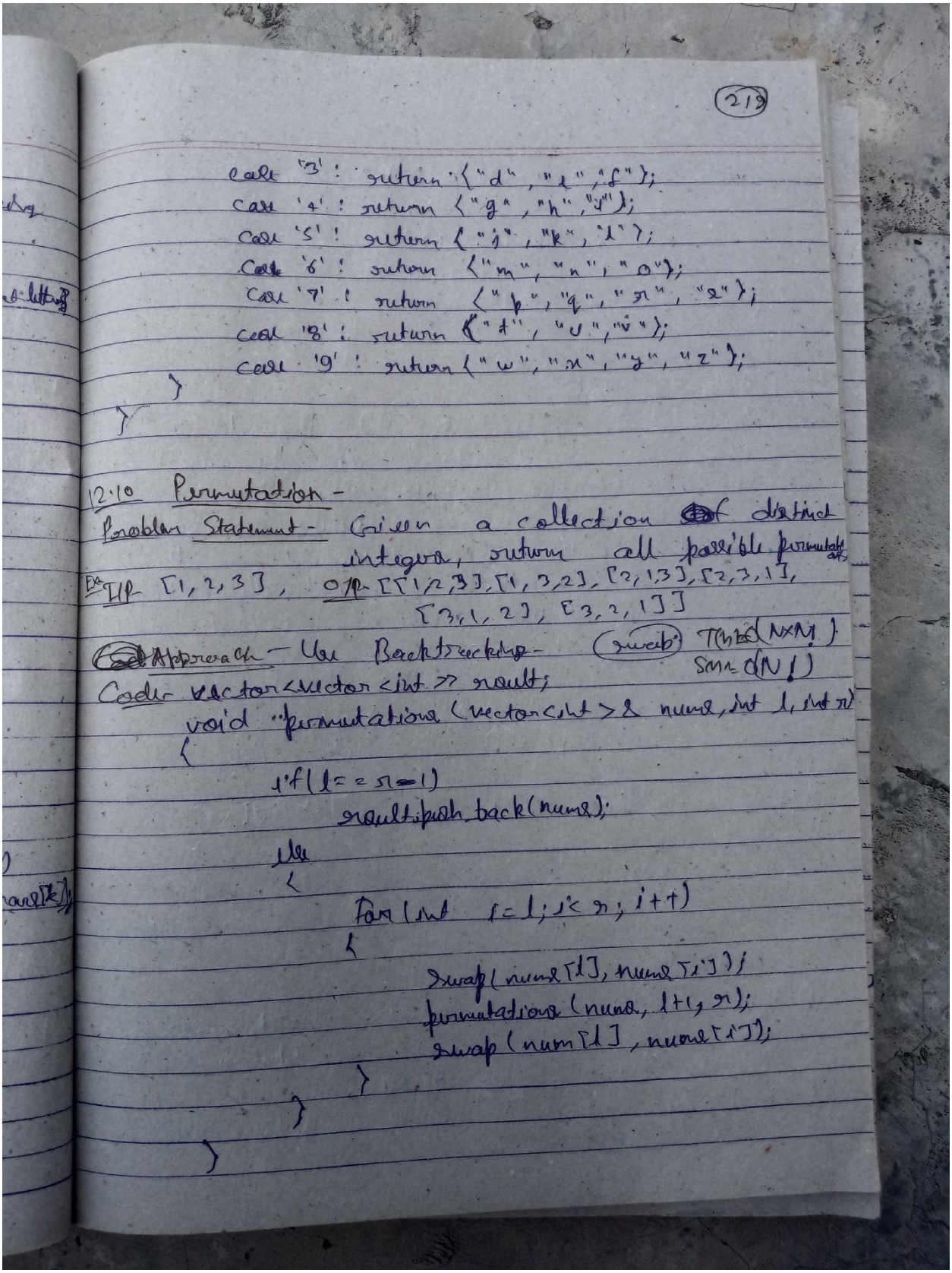


Approach - Use Backtracking - $T(N) = O(3^N 4^M)$ $S_N = O(3^N \times 4^M)$

Algo - ① If there is no more digits to check that means the current combination is completed.

② If there are still digits to check -


218
(i) Iterate over the letters mapping the next available digit.
(ii) Append the current letter to current combination [combination + current letter]
(iii) Proceed to check next digits (combination, letters, next_digits[1:]).
Code vector<string> letterCombinations(string digits)
{
vector<string> combo;
vector<string> chars;
if(digits.size()==0)
return combo;
combo=getChars(digits[0]);
for(int i=1; i<digits.size();i++)
{
vector<string> temp=combo;
chars=get(digits[i]);
combo.clear();
for(int j=0; j<temp.size(); j++)
for(int k=0; k<chars.size(); k++)
combo.push_back(temp[j]+chars[k]
}
return combo;
}
vector<string> getChars(char num)
{
switch(num)
{
case '2': return {"a", "b", "c"};



219

```
case '3' : return {"d", "e", "f"};
case '4' : return {"g", "h", "i"};
case '5' : return {"j", "k", "l"};
case '6' : return {"m", "n", "o"};
case '7' : return {"p", "q", "r", "s"};
case '8' : return {"t", "u", "v"};
case '9' : return {"w", "x", "y", "z"};
```

12.10 Permutation -

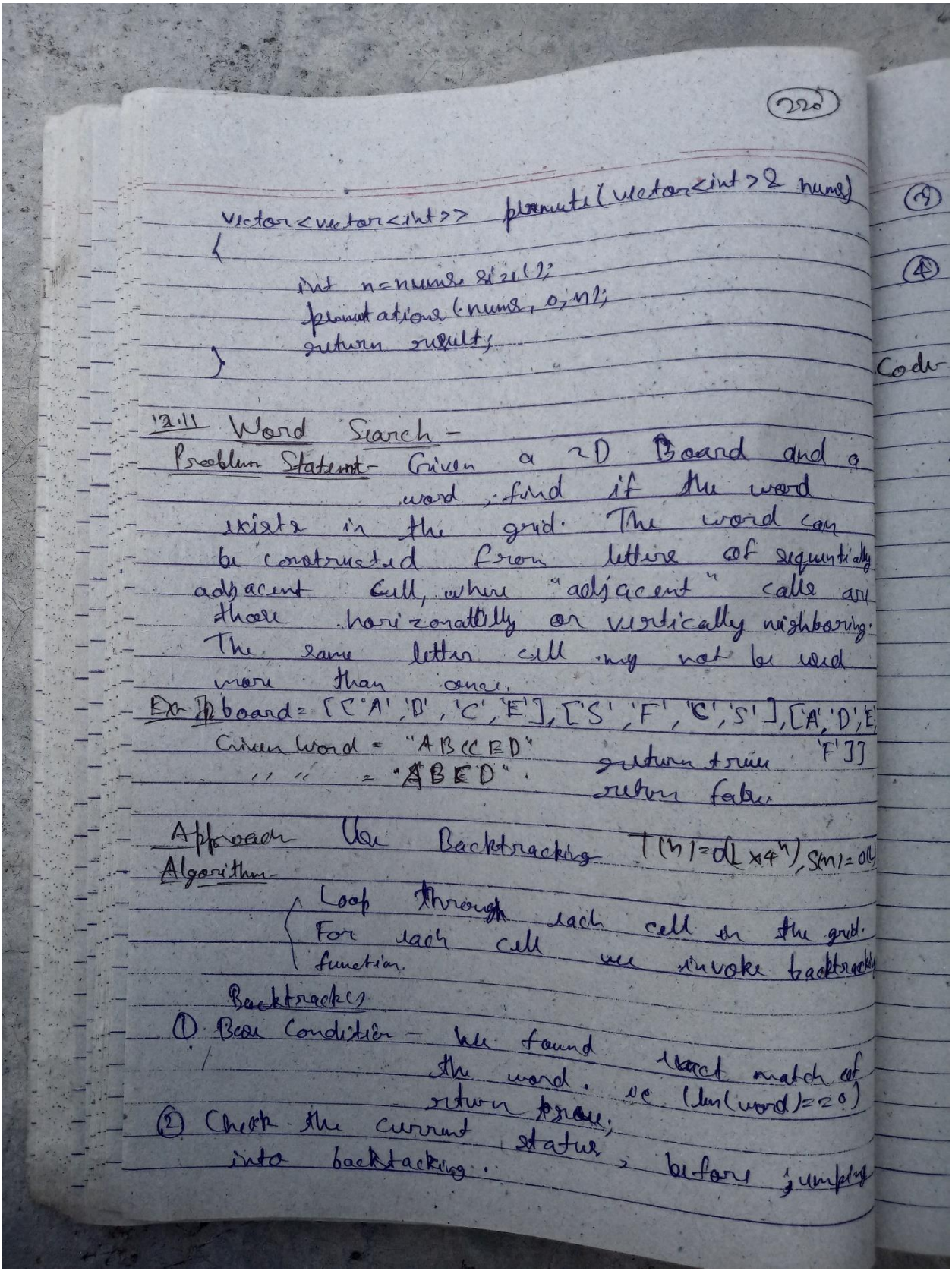
Problem Statement - Given a collection of distinct integers, return all possible permutations.

Ex- I/P [1, 2, 3], O/P- [[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]

Approach - Use Backtracking - (recursion) T(n) = O(N×N!) SC = O(N!)

Code-

```
vector<vector<int>> result;
void permutations(vector<int>& nums, int l, int r)
{
    if(l==r-1)
        result.push_back(nums);
    else
    {
        for(int i=l; i<r; i++)
        {
            swap(nums[l], nums[i]);
            permutations(nums, l+1, r);
            swap(nums[l], nums[i]);
        }
    }
}
```



(220)

```
vector<vector<int>> permute(vector<int>& nums)
{
    int n=nums.size();
    permutation(nums, 0, n);
    return result;
}
```

Code

12.11 Word Search -

Problem Statement - Given a 2D Board and a word, find if the word exists in the grid. The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

Ex- board = [['A','B','C','E'],['S','F','C','S'],['A','D','E','F']]
Given word = "ABCCED" return true
" " = "ABED". return false

Approach Use Backtracking $T(n) = O(L \times 4^n)$, $S(n) = O(L)$

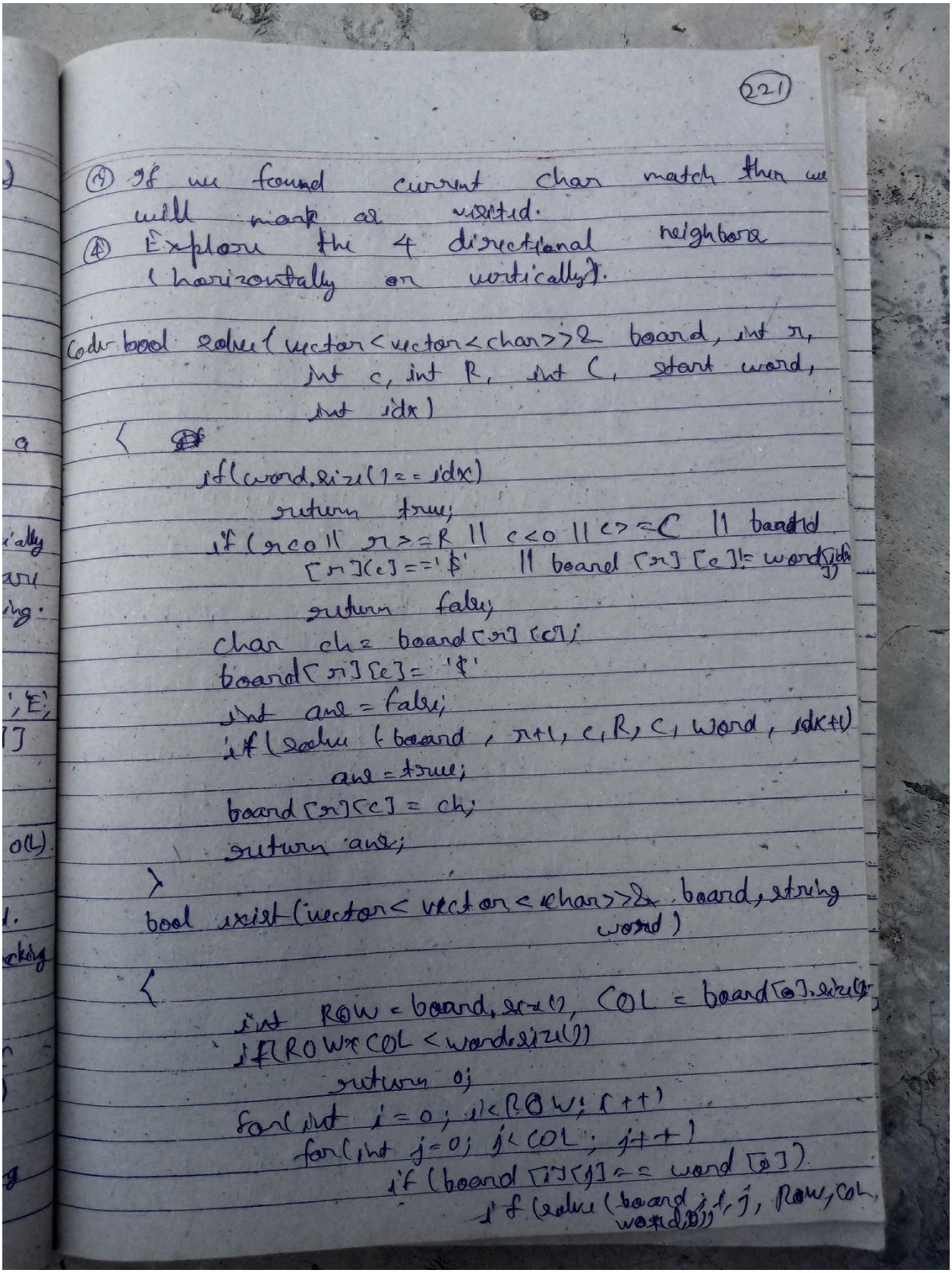
Algorithm -

Loop through each cell in the grid.
For each cell we invoke backtracking function.

Backtracking

① Base Condition - We found exact match of the word. i.e (len(word)==0) return true;

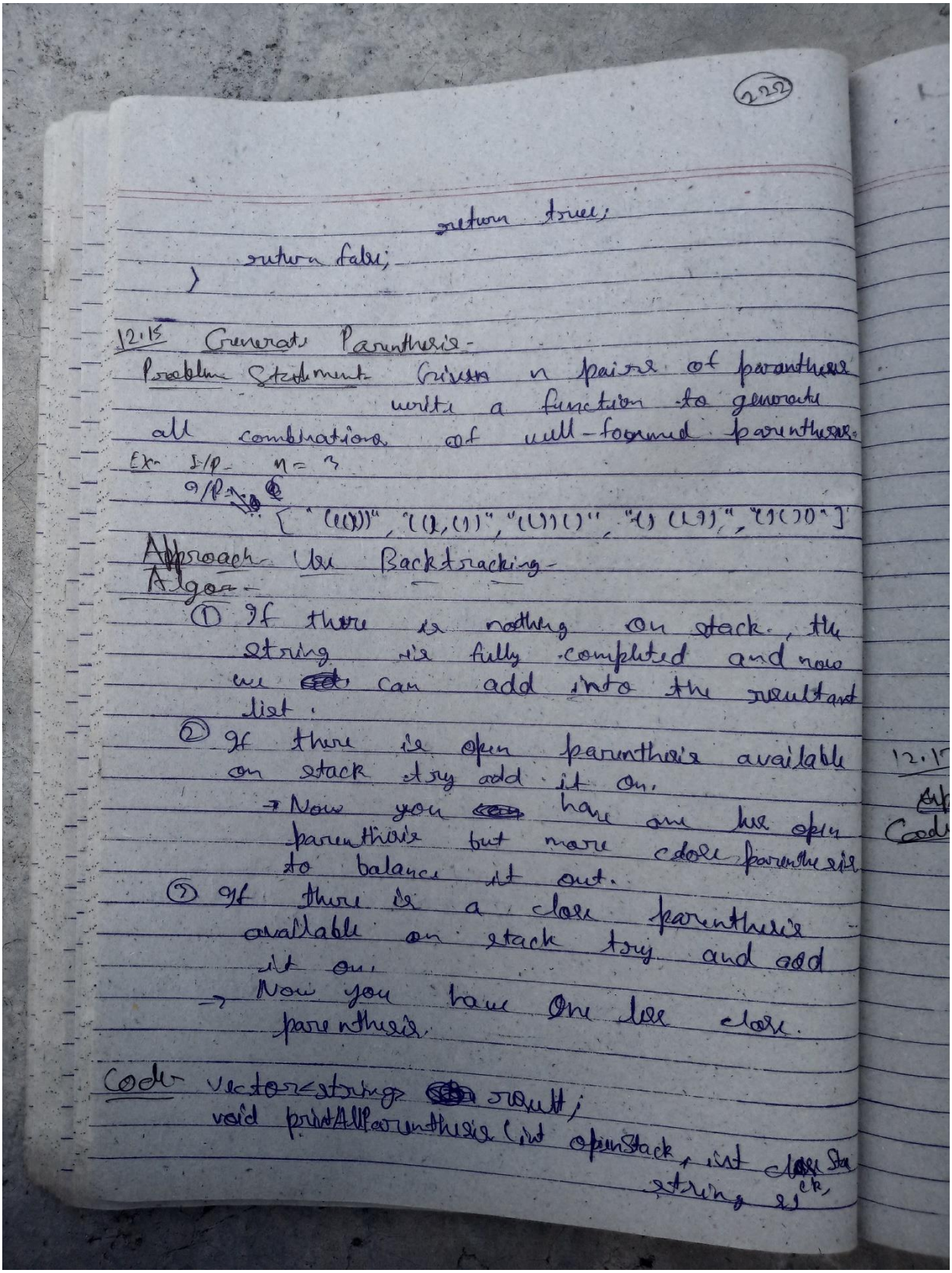
② Check the current status, before jumping into backtracking.



221

③ If we found current char match then we will mark as visited.
④ Explore the 4 directional neighbors (horizontally or vertically).

```
bool solve(vector<vector<char>>& board, int r,
           int c, int R, int C, string word,
           int idx)
{
    if(word.size()==idx)
        return true;
    if(r<0 || r>=R || c<0 || c>=C || board
       [r][c]=='$' || board[r][c]!=word[idx])
        return false;
    char ch = board[r][c];
    board[r][c] = '$';
    int ans = false;
    if(solve(board, r+1, c, R, C, word, idx+1))
        ans = true;
    board[r][c] = ch;
    return ans;
}
bool exist(vector<vector<char>>& board, string
           word)
{
    int ROW = board.size(), COL = board[0].size();
    if(ROW*COL < word.size())
        return 0;
    for(int i=0; i<ROW; i++)
        for(int j=0; j<COL; j++)
            if(board[i][j]==word[0])
                if(solve(board, i, j, ROW, COL,
                         word))
```



222

return true;

return false;

}

12.15 Generate Parenthesis-

Problem Statement- Given n pairs of parentheses write a function to generate all combinations of well-formed parentheses.

Ex- I/P- n = 3

O/P- ["((()))", "(()())", "(())()", "()(())", "()()()"]

Approach- Use Backtracking-

Algo-

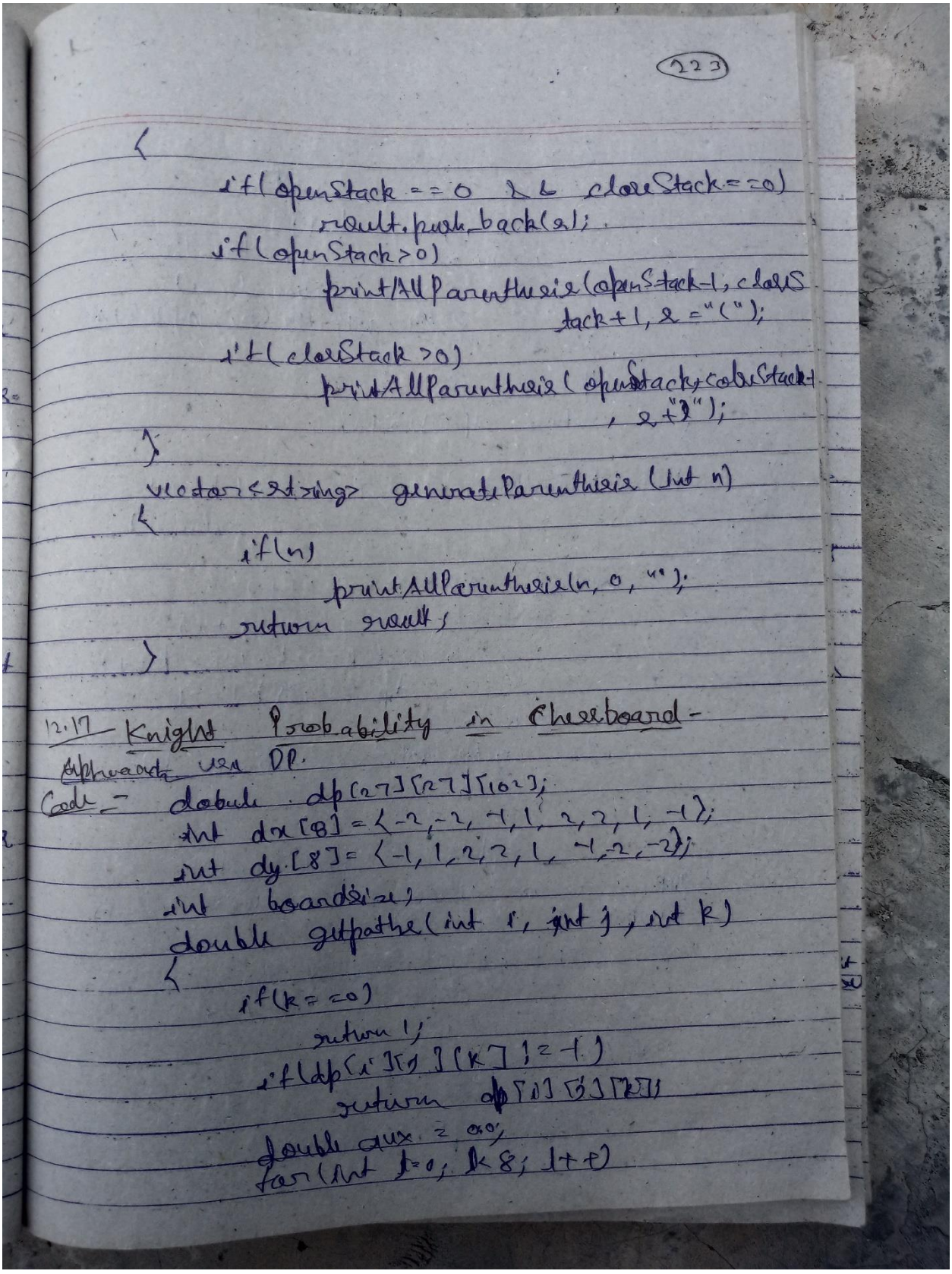
① If there is nothing on stack, the string is fully completed and now we can add into the resultant list.

② If there is open parenthesis available on stack try add it on.
→ Now you have one less open parenthesis but more close parenthesis to balance it out.

③ If there is a close parenthesis available on stack try and add it on.
→ Now you have one less close parenthesis.

Code- vector<string> result;
void printAllParenthesis(int openStack, int closeStack, string s)

12.17
Algo
Code



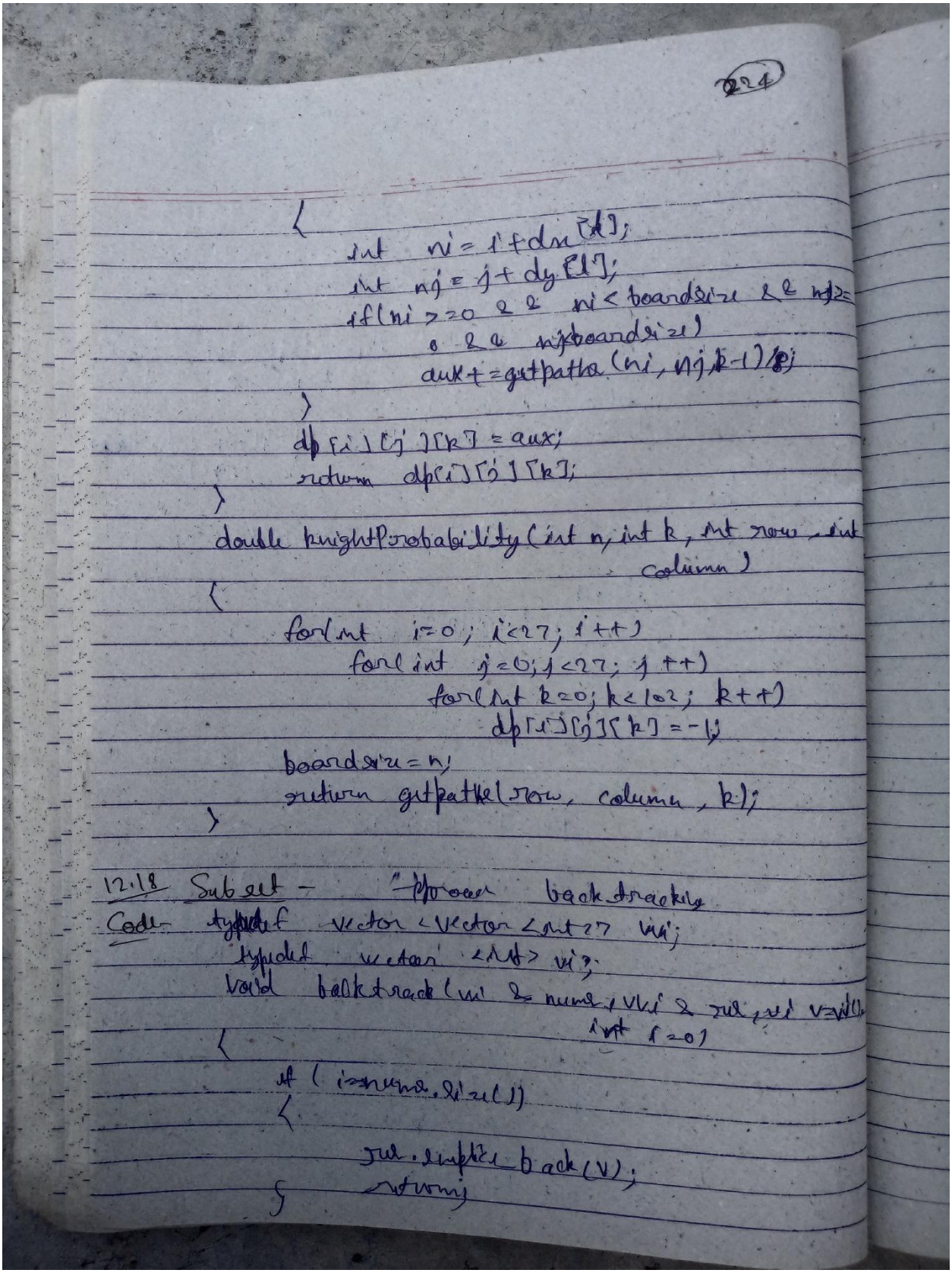
223

```
{
    if(openStack == 0 && closeStack == 0)
        result.push_back(s);
    if(openStack > 0)
        printAllParenthesis(openStack-1, closeStack+1, s+"(");
    if(closeStack > 0)
        printAllParenthesis(openStack, closeStack-1, s+")");
}
vector<string> generateParenthesis(int n)
{
    if(n)
        printAllParenthesis(n, 0, "");
    return result;
}
```

12.17 Knight Probability in Chessboard -
Approach use DP.

Code -

```
double dp[27][27][102];
int dx[8] = {-2,-2,-1,1,2,2,1,-1};
int dy[8] = {-1,1,2,2,1,-1,-2,-2};
int boardSize;
double getPath(int i, int j, int k)
{
    if(k == 0)
        return 1;
    if(dp[i][j][k] != -1)
        return dp[i][j][k];
    double aux = 0.0;
    for(int l=0; l<8; l++)
```



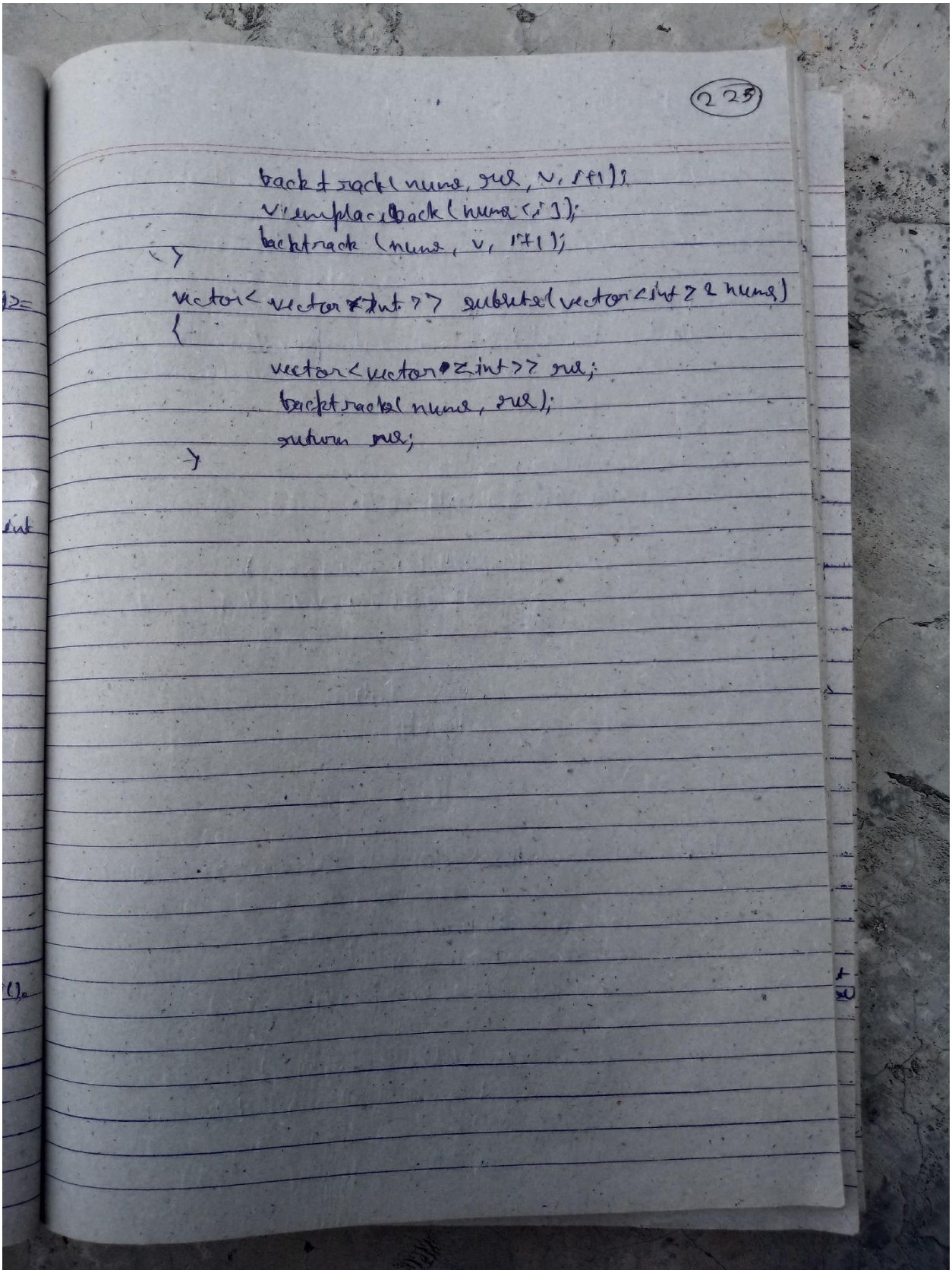
224

```
{
    int ni = i + dx[k];
    int nj = j + dy[k];
    if(ni >= 0 && ni < boardsize && nj >=
        0 && nj < boardsize)
        aux += getpath(ni, nj, k-1);
}
dp[i][j][k] = aux;
return dp[i][j][k];
}

double knightProbability(int n, int k, int row, int
                         column)
{
    for(int i=0; i<27; i++)
        for(int j=0; j<27; j++)
            for(int k=0; k<102; k++)
                dp[i][j][k] = -1;
    boardsize = n;
    return getpath(row, column, k);
}
```

12.18 Subset - Power backtracking

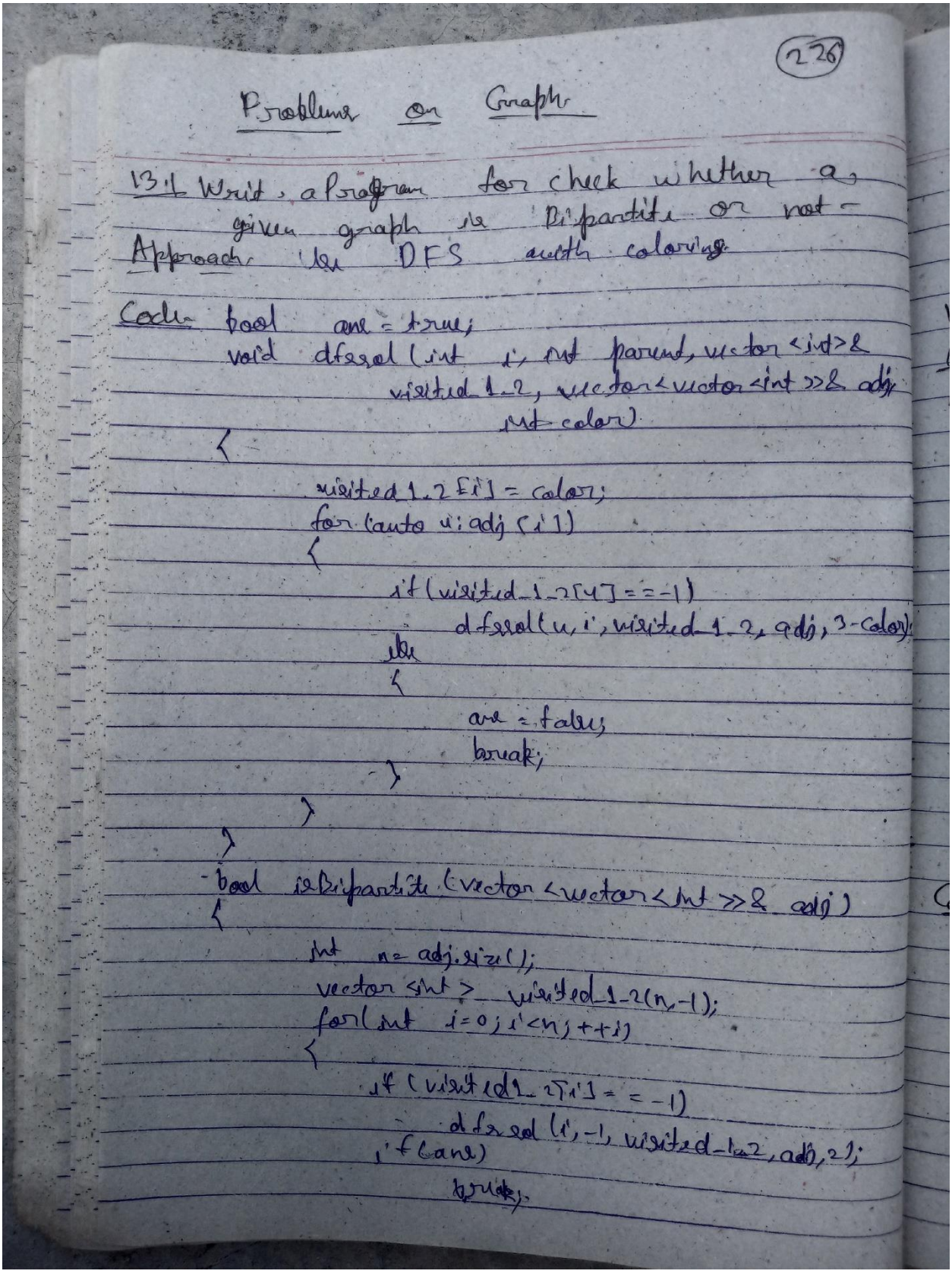
Code -

```
typedef vector<vector<int>27> vii;
typedef vector<int> vi;
void backtrack(vi & nums, vvi & res, vi & v, int i=0)
{
    if (i==nums.size())
    {
        res.push_back(v);
        return;
    }
```



225

```
        backtrack(nums, res, v, i+1);
        v.emplace_back(nums[i]);
        backtrack(nums, v, i+1);
    }
vector<vector<int>> subsets(vector<int>& nums)
{
    vector<vector<int>> res;
    backtrack(nums, res);
    return res;
}
```

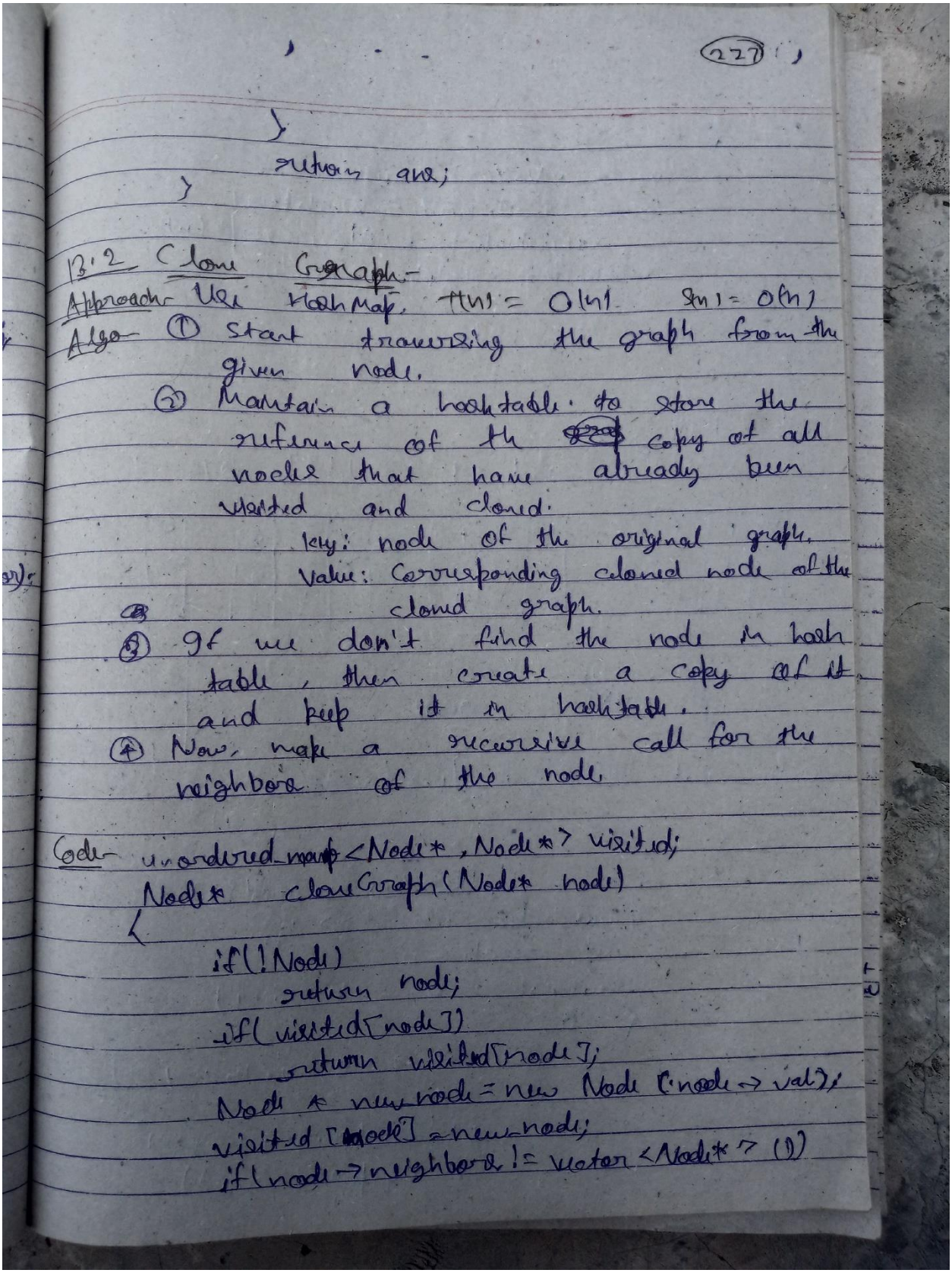


226

Problems on Graph

13. Write a Program for check whether a given graph is Bipartite or not -
Approach: Use DFS with coloring.

Code:
```
bool ans = true;
void dfsrel(int i, int parent, vector<int>& visited_1_2, vector<vector<int>>& adj, int color)
{
    visited_1_2[i] = color;
    for(auto u: adj[i])
    {
        if(visited_1_2[u] == -1)
            dfsrel(u, i, visited_1_2, adj, 3-color);
        else
        {
            ans = false;
            break;
        }
    }
}
bool isBipartite(vector<vector<int>>& adj)
{
    int n = adj.size();
    vector<int> visited_1_2(n, -1);
    for(int i=0; i<n; ++i)
    {
        if(visited_1_2[i] == -1)
            dfsrel(i, -1, visited_1_2, adj, 2);
        if(!ans)
            break;
```



227

```
    }
    return ans;
}
```

B.2 Clone Graph.-

Approach- Use HashMap. TM = O(n) SM = O(n)

Algo- ① Start traversing the graph from the given node.

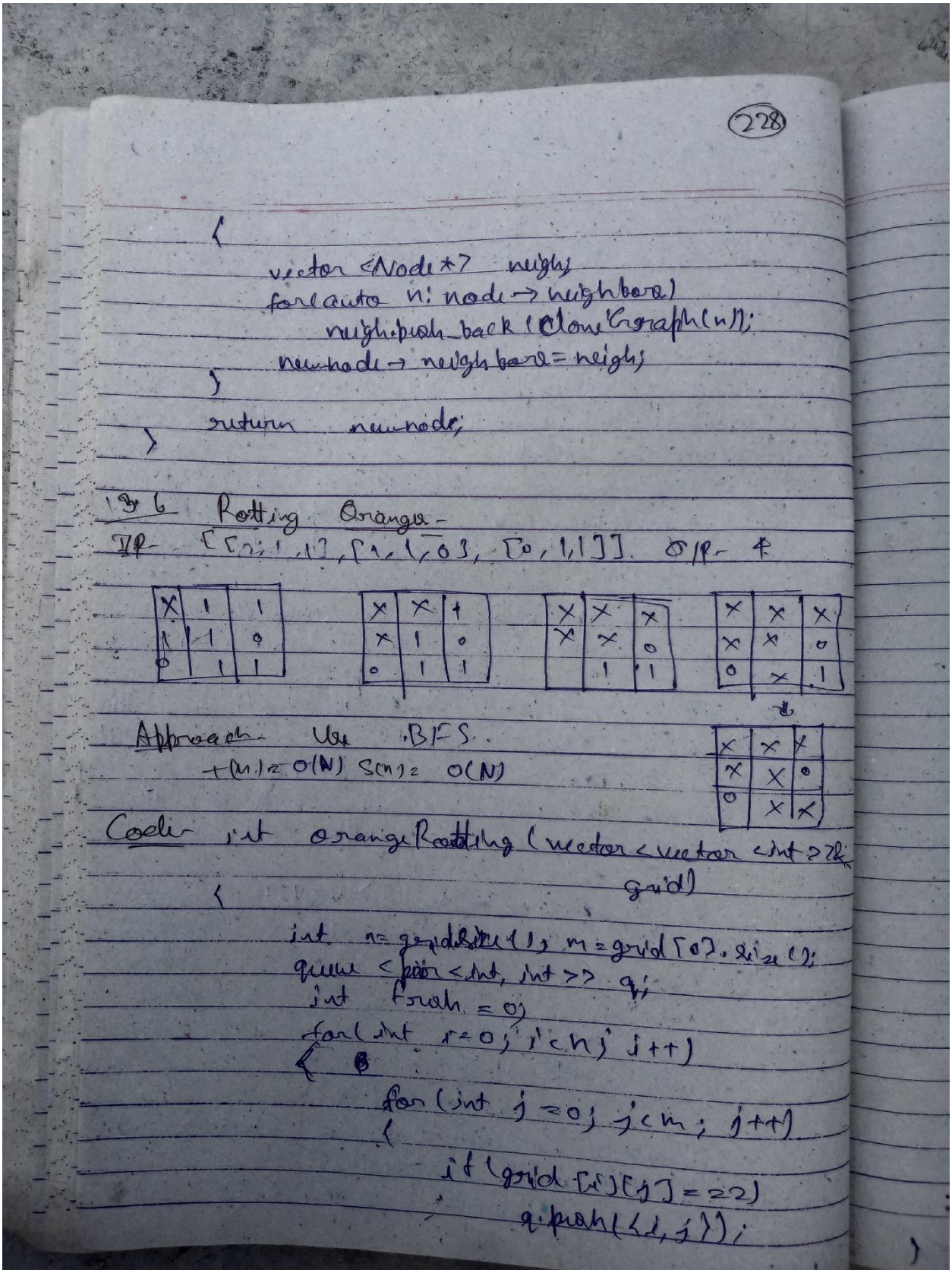
② Maintain a hashtable to store the reference of the copy of all nodes that have already been visited and cloned.
key: node of the original graph.
Value: Corresponding cloned node of the cloned graph.

③ If we don't find the node in hash table, then create a copy of it and keep it in hashtable.

④ Now, make a recursive call for the neighbors of the node.

Code-

```
unordered_map<Node*, Node*> visited;
Node* cloneGraph(Node* node)
{
    if(!Node)
        return node;
    if(visited[node])
        return visited[node];
    Node* newnode = new Node(node->val);
    visited[node] = newnode;
    if(node->neighbors != vector<Node*>())
```



228

```
{
    vector<Node*> neigh;
    for(auto n: node->neighbors)
        neigh.push_back(cloneGraph(n));
    newnode->neighbors = neigh;
}
return newnode;
}
```

Q6 Rotting Oranges -

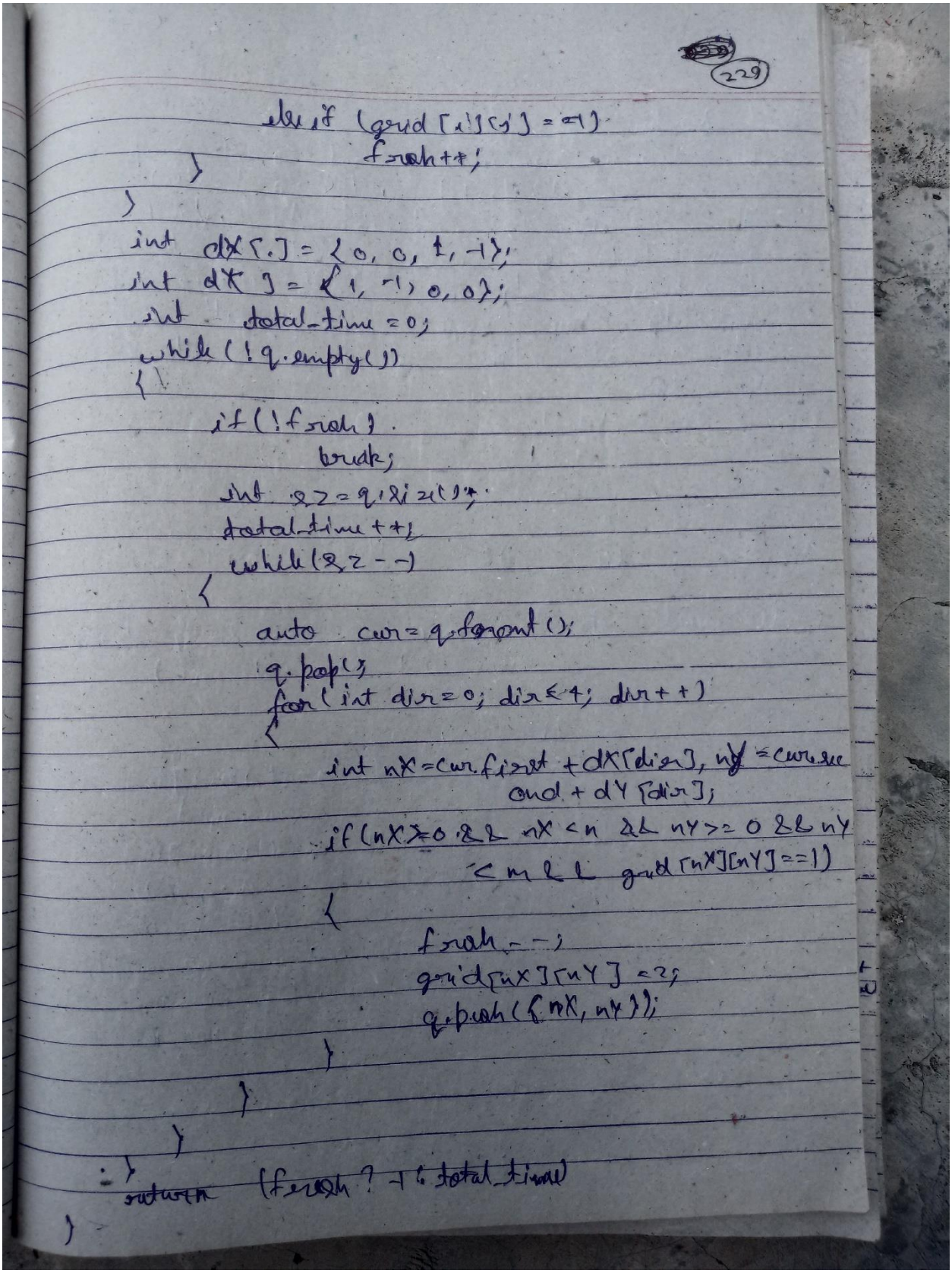
I/P - [[2,1,1],[1,1,0],[0,1,1]] O/P - 4

Approach - Use BFS.

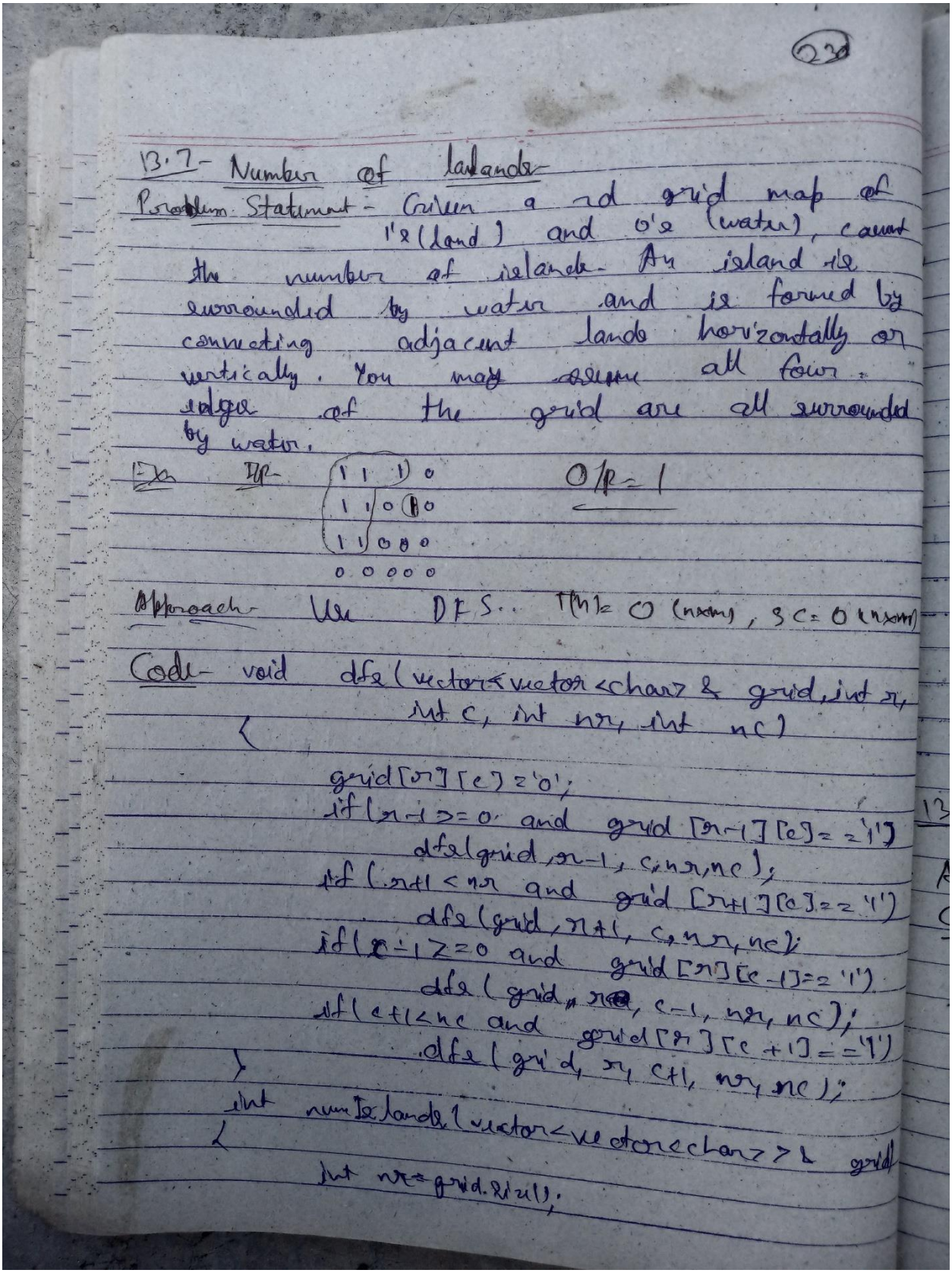
T(n) = O(N) S(n) = O(N)

Code

```
int orangeRotting(vector<vector<int>>& grid)
{
    int n = grid.size(), m = grid[0].size();
    queue<pair<int,int>> q;
    int fresh = 0;
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<m; j++)
        {
            if(grid[i][j]==2)
                q.push({i,j});
```



```
229

        else if (grid[i][j] == 1)
            fresh++;
    }
}
int dX[.] = {0, 0, 1, -1};
int dY[] = {1, -1, 0, 0};
int total_time = 0;
while (!q.empty())
{
    if (!fresh)
        break;
    int sz = q.size();
    total_time++;
    while (sz--)
    {
        auto cur = q.front();
        q.pop();
        for (int dir = 0; dir < 4; dir++)
        {
            int nX = cur.first + dX[dir], nY = cur.se
                cond + dY[dir];
            if (nX >= 0 && nX < n && nY >= 0 && nY
                < m && grid[nX][nY] == 1)
            {
                fresh--;
                grid[nX][nY] = 2;
                q.push({nX, nY});
            }
        }
    }
}
return (fresh ? -1 : total_time);
}
```



230

13.7- Number of Islands

Problem Statement - Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

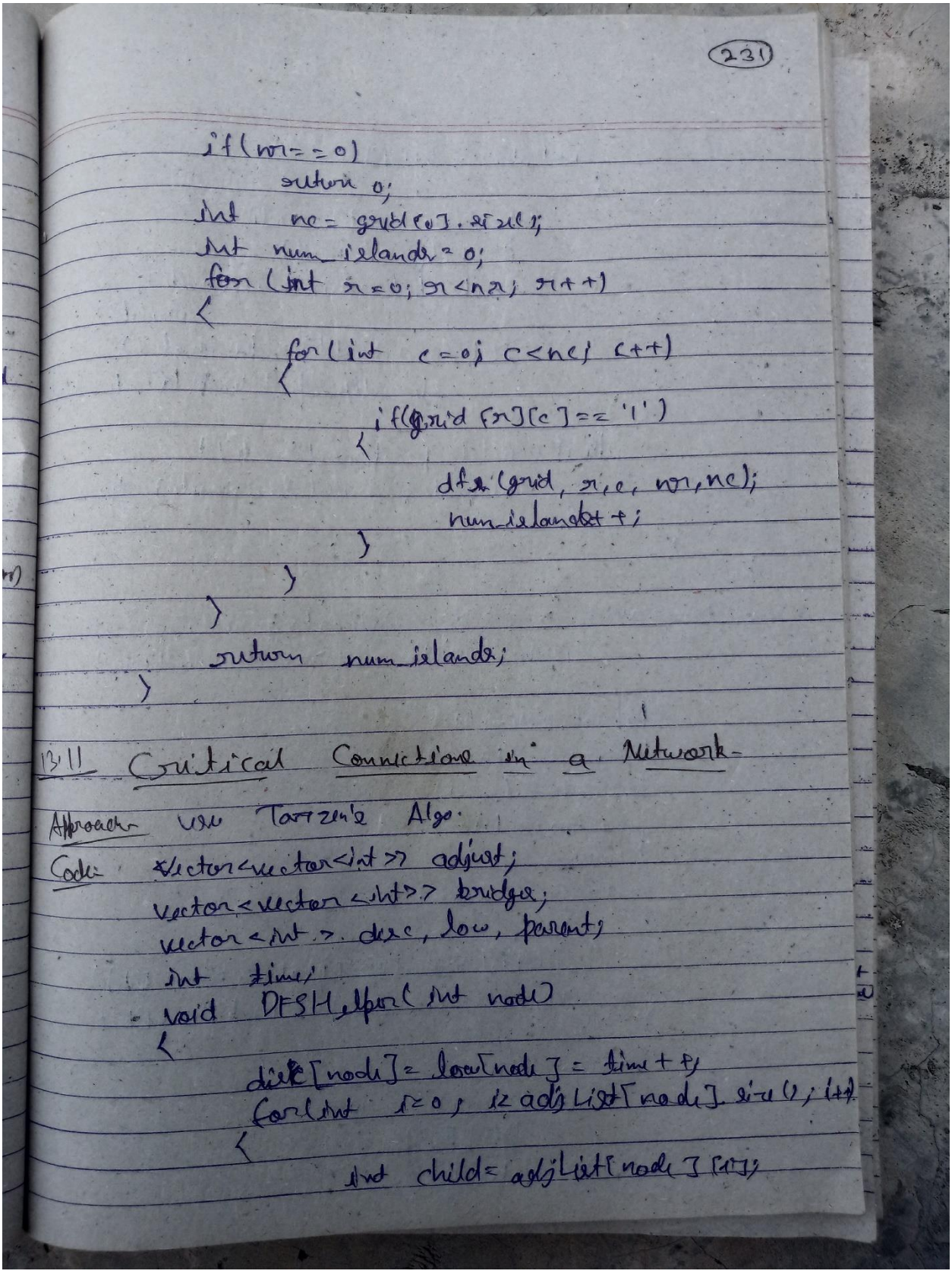
Ex- Ip- 1 1 1 1 0
1 1 0 1 0
1 1 0 0 0
0 0 0 0 0

O/p = 1

Approach- Use DFS.. T(n) = O(n×m), S.C = O(n×m)

Code-

```
void dfs(vector<vector<char>> & grid, int r,
         int c, int nr, int nc)
{
    grid[r][c] = '0';
    if(r-1 >= 0 and grid[r-1][c] == '1')
        dfs(grid, r-1, c, nr, nc);
    if(r+1 < nr and grid[r+1][c] == '1')
        dfs(grid, r+1, c, nr, nc);
    if(c-1 >= 0 and grid[r][c-1] == '1')
        dfs(grid, r, c-1, nr, nc);
    if(c+1 < nc and grid[r][c+1] == '1')
        dfs(grid, r, c+1, nr, nc);
}
int numIslands(vector<vector<char>>& grid)
{
    int nr = grid.size();
```



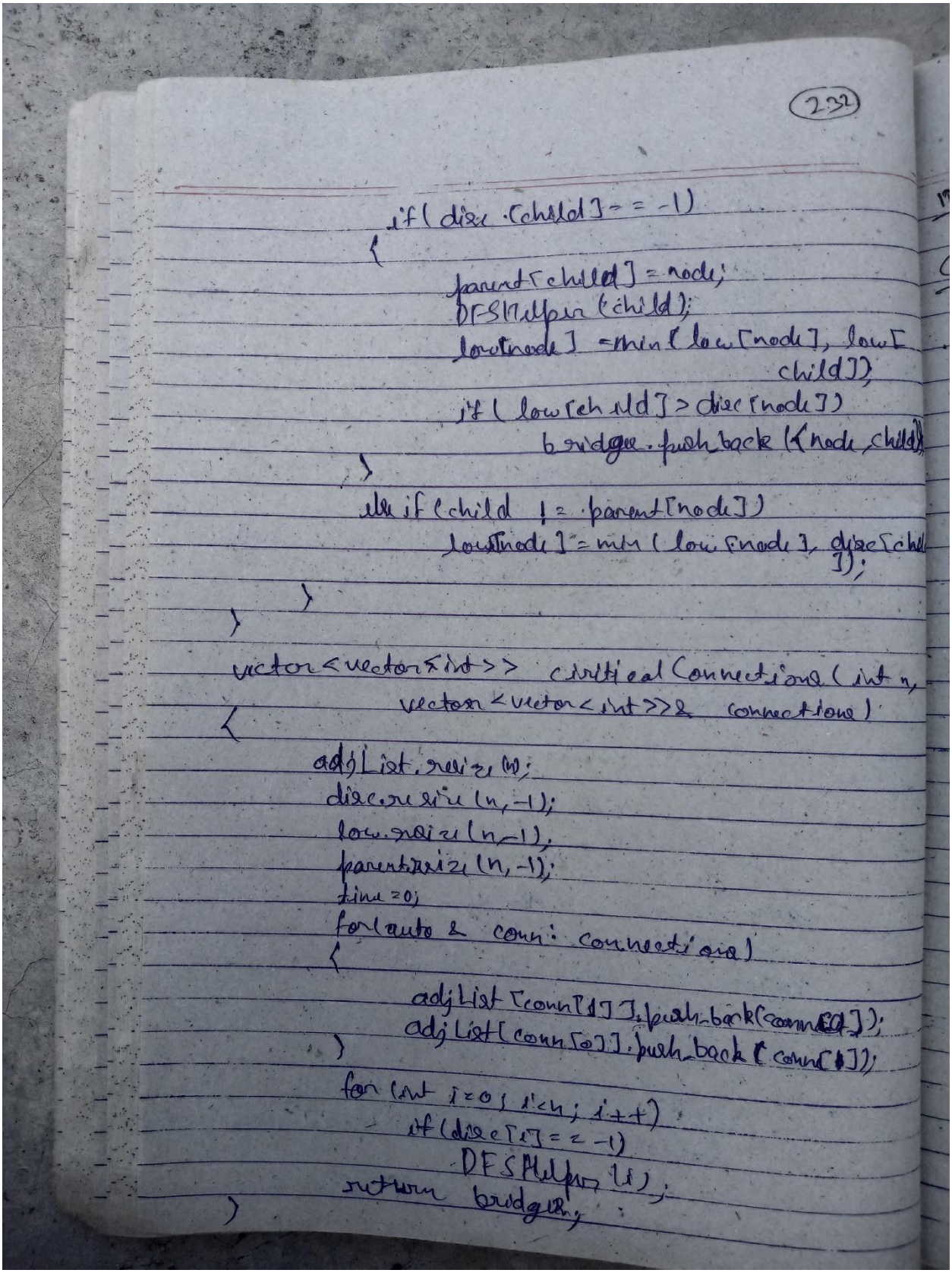
231

```
if(nr==0)
    return 0;
int nc = grid[0].size();
int num_islands = 0;
for (int r=0; r<nr; r++)
{
    for(int c=0; c<nc; c++)
    {
        if(grid[r][c]=='1')
        {
            dfs(grid, r, c, nr, nc);
            num_islands++;
        }
    }
}
return num_islands;
}
```

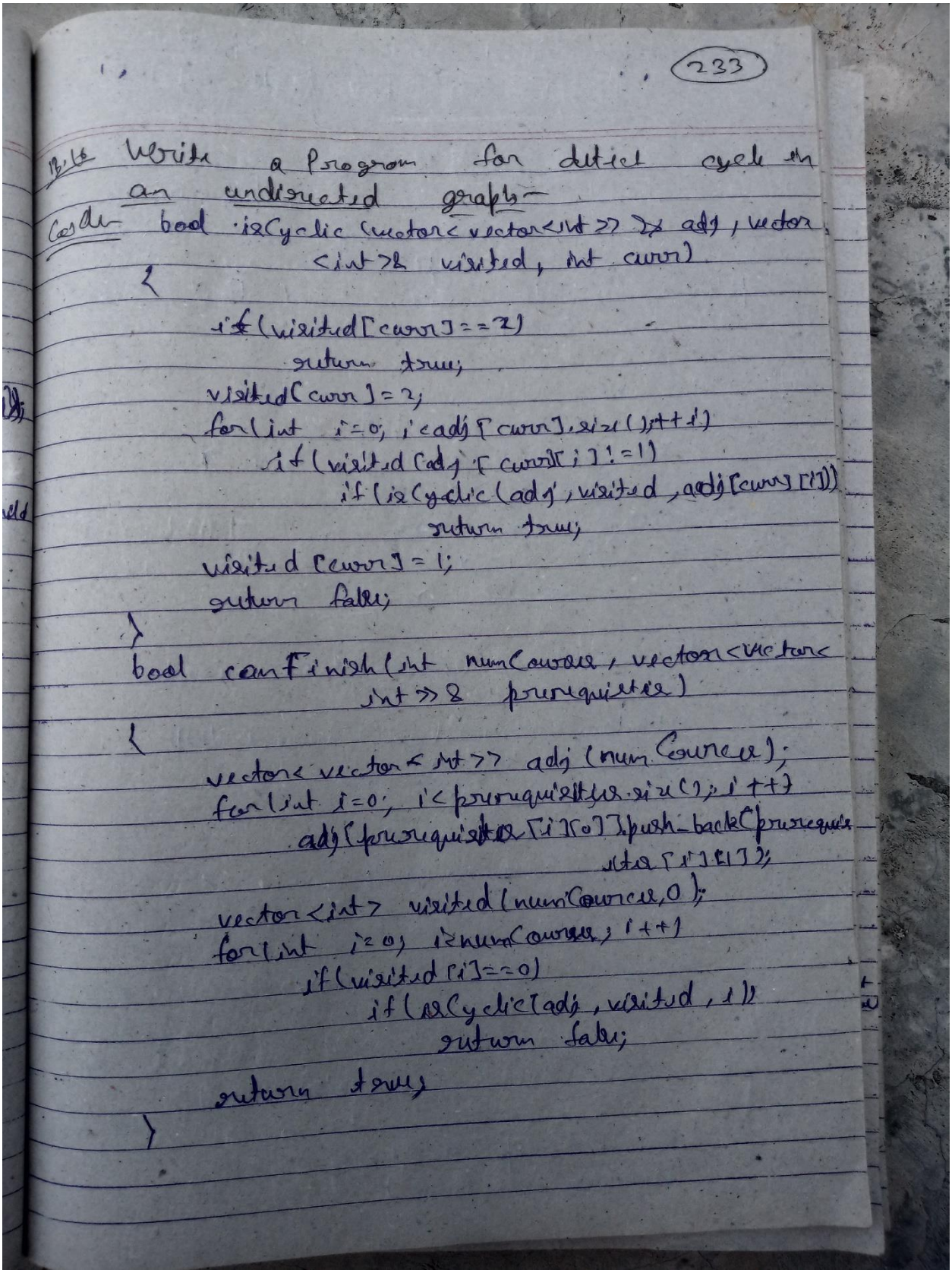
1311 Critical Connections in a Network-

Approach use Tarzan's Algo.

Code: vector<vector<int>> adjust;
vector<vector<int>> bridges;
vector<int> disc, low, parent;
int time;
void DFSHelper(int node)
{
    disc[node] = low[node] = time++;
    for(int i=0; i<adjList[node].size(); i++)
    {
        int child = adjList[node][i];



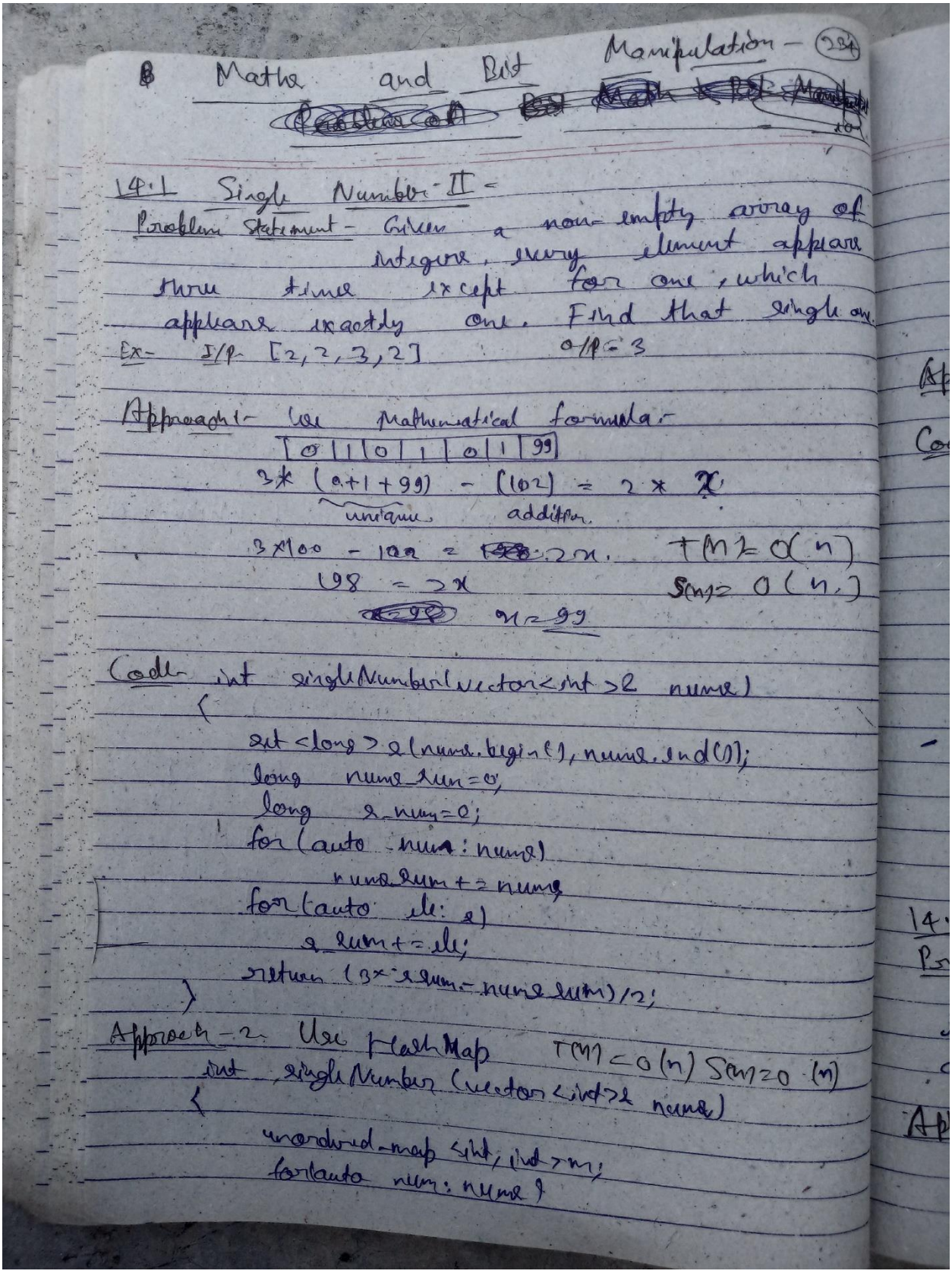
```
(232)

        if (disc[child] == -1)
        {
            parent[child] = node;
            DFSHelper(child);
            low[node] = min(low[node], low[
                                child]);
            if (low[child] > disc[node])
                bridges.push_back({node, child});
        }
        else if (child != parent[node])
            low[node] = min(low[node], disc[child
                                ]);
    }
}

vector<vector<int>> criticalConnections(int n,
        vector<vector<int>>& connections)
{
    adjList.resize(n);
    disc.resize(n, -1);
    low.resize(n, -1);
    parent.resize(n, -1);
    time = 0;
    for(auto & conn : connections)
    {
        adjList[conn[1]].push_back(conn[0]);
        adjList[conn[0]].push_back(conn[1]);
    }
    for (int i=0; i<n; i++)
        if (disc[i] == -1)
            DFSHelper(i);
    return bridges;
}
```



233

Q. Write a Program for detect cycle in an undirected graph-

```
Code- bool isCyclic(vector<vector<int>> &adj, vector
          <int> &visited, int curr)
{
    if(visited[curr]==2)
        return true;
    visited[curr]=2;
    for(int i=0; i<adj[curr].size();++i)
        if(visited[adj[curr][i]]!=1)
            if(isCyclic(adj, visited, adj[curr][i]))
                return true;
    visited[curr]=1;
    return false;
}
bool canFinish(int numCourses, vector<vector<
          int>>& prerequisites)
{
    vector<vector<int>> adj(numCourses);
    for(int i=0; i<prerequisites.size();i++)
        adj[prerequisites[i][0]].push_back(prerequis
                                    ites[i][1]);
    vector<int> visited(numCourses,0);
    for(int i=0; i<numCourses; i++)
        if(visited[i]==0)
            if(isCyclic(adj, visited, i))
                return false;
    return true;
}
```



8 Maths and Bit Manipulation – (234)

14.1 Single Number - II =

Problem Statement - Given a non-empty array of integers, every element appears three times except for one, which appears exactly one. Find that single one.

Ex- I/P- [2, 2, 3, 2] O/P = 3

Approach:- Use mathematical formula:-

[0 | 1 | 0 | 1 | 0 | 1 | 99]

3 * (0+1+99) - [102] = 2 * x

unique addition

3*100 - 102 = 2x. T(n) = O(n)

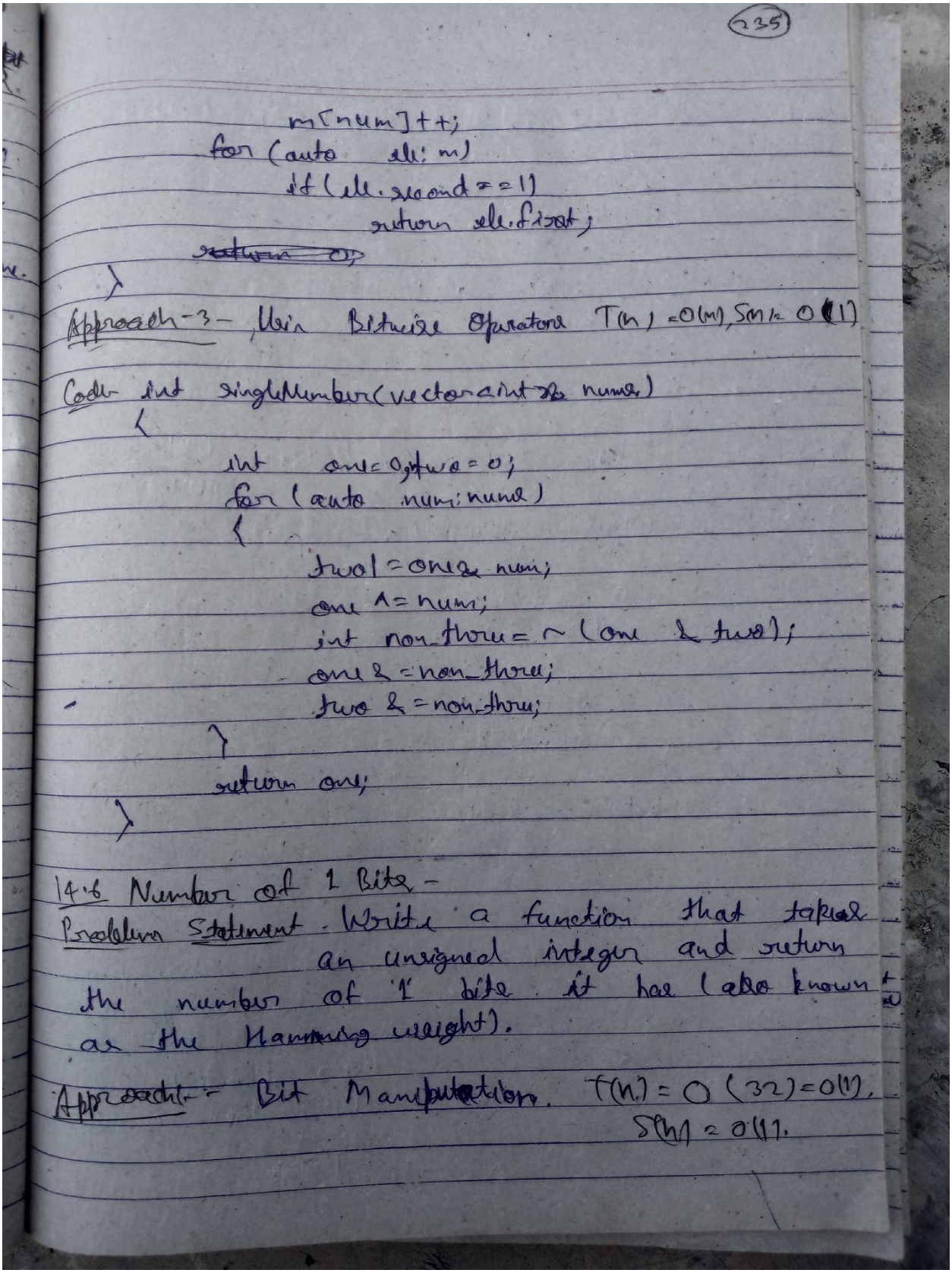
198 = 2x S(n) = O(n)

x = 99

Code- int singleNumber(vector<int>& nums)
{
set<long> s(nums.begin(), nums.end());
long nums_sum = 0;
long s_sum = 0;
for(auto num: nums)
nums_sum += num;
for(auto it: s)
s_sum += it;
return (3*s_sum - nums_sum)/2;
}

Approach - 2: Use HashMap T(n) = O(n) S(n) = O(n)

int singleNumber(vector<int>& nums)
{
unordered_map<int, int> m;
for(auto num: nums)



235

```
        m[num]++;
    for (auto ele: m)
        if (ele.second == 1)
            return ele.first;
    return 0;
}
```

Approach-3 - Using Bitwise Operators T(n) = O(n), S(n) = O(1)

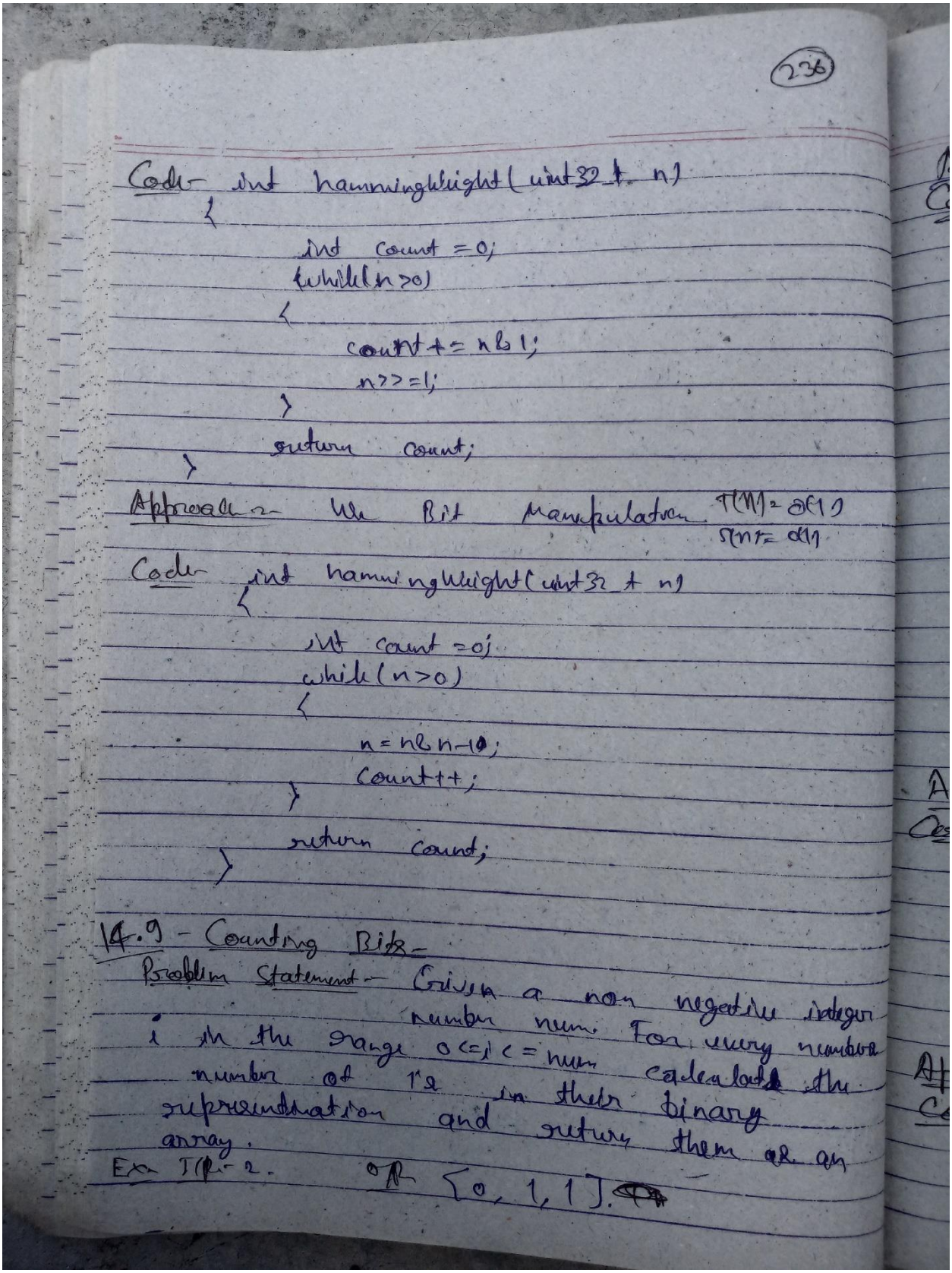
Code

```
int singleNumber(vector<int>& nums)
{
    int one=0, two=0;
    for (auto num: nums)
    {
        two |= one & num;
        one ^= num;
        int non_three = ~(one & two);
        one &= non_three;
        two &= non_three;
    }
    return one;
}
```

14. Number of 1 Bits -

Problem Statement - Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the Hamming weight).

Approach - Bit Manipulation. T(n) = O(32) = O(1).
S(n) = O(1).



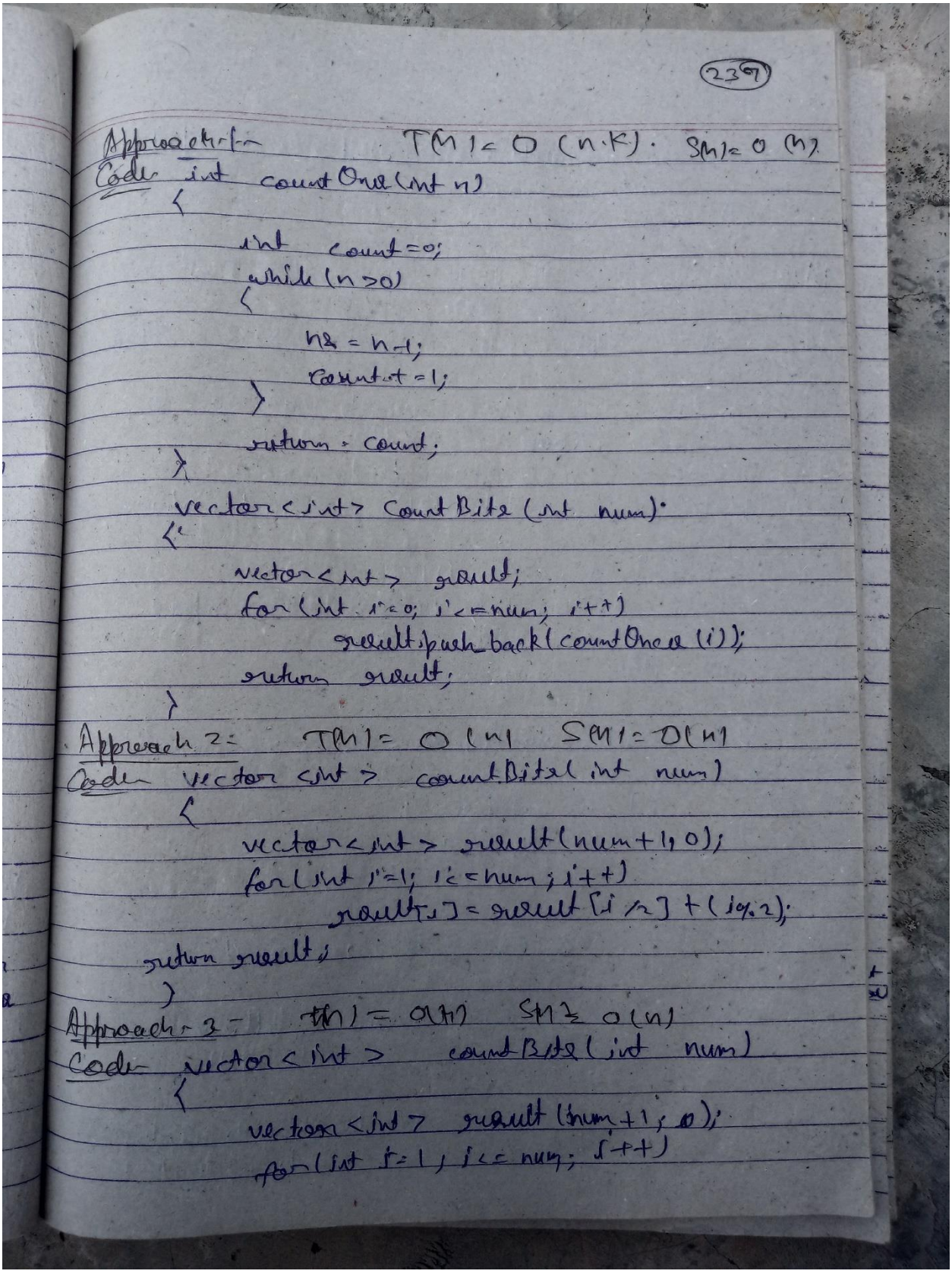
236

Code- int hammingWeight(uint32_t n)
{
int count = 0;
while(n>0)
{
count += n&1;
n>>=1;
}
return count;
}

Approach:- We Bit manipulation T(n) = O(1)
S(n) = O(1)

Code- int hammingWeight(uint32_t n)
{
int count = 0;
while(n>0)
{
n = n&n-1;
count++;
}
return count;
}

14.9 - Counting Bits-
Problem Statement- Given a non negative integer number num. For every numbers i in the range 0<=i<=num calculate the number of 1's in their binary representation and return them as an array.
Ex- I/P- 2. O/P [0, 1, 1].



(239)

Approach-1 :- T(n) = O(n·K) · S(n) = O(n)

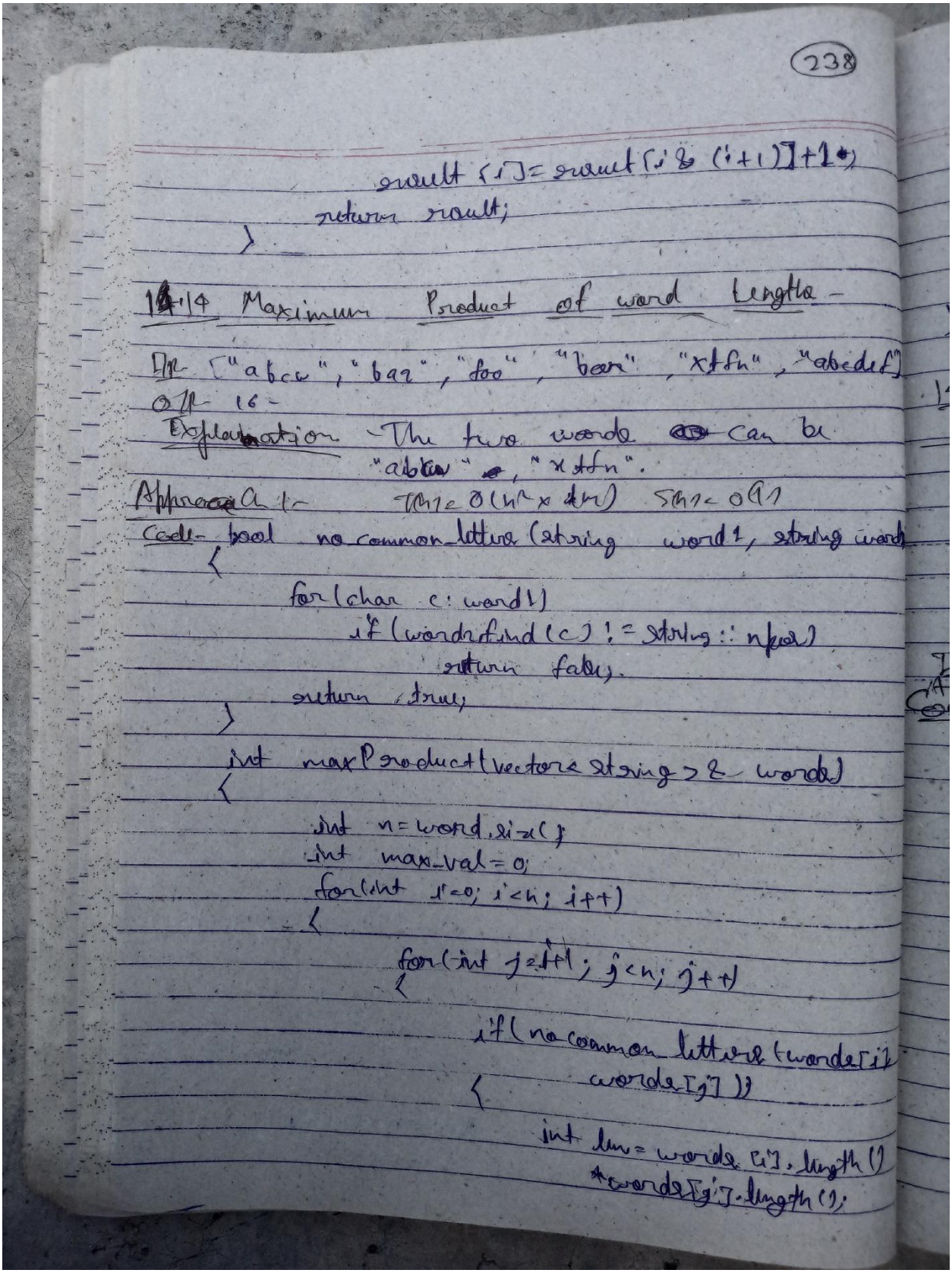
Code
```
int countOne(int n)
{
    int count=0;
    while(n>0)
    {
        n& = n-1;
        count++;
    }
    return count;
}
vector<int> countBits(int num)
{
    vector<int> result;
    for(int i=0; i<=num; i++)
        result.push_back(countOne(i));
    return result;
}
```

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

Code
```
vector<int> countBits(int num)
{
    vector<int> result(num+1, 0);
    for(int i=1; i<=num; i++)
        result[i] = result[i/2] + (i%2);
    return result;
}
```

Approach-3 :- T(n) = O(n) S(n) = O(n)

Code
```
vector<int> countBits(int num)
{
    vector<int> result(num+1, 0);
    for(int i=1; i<=num; i++)
```

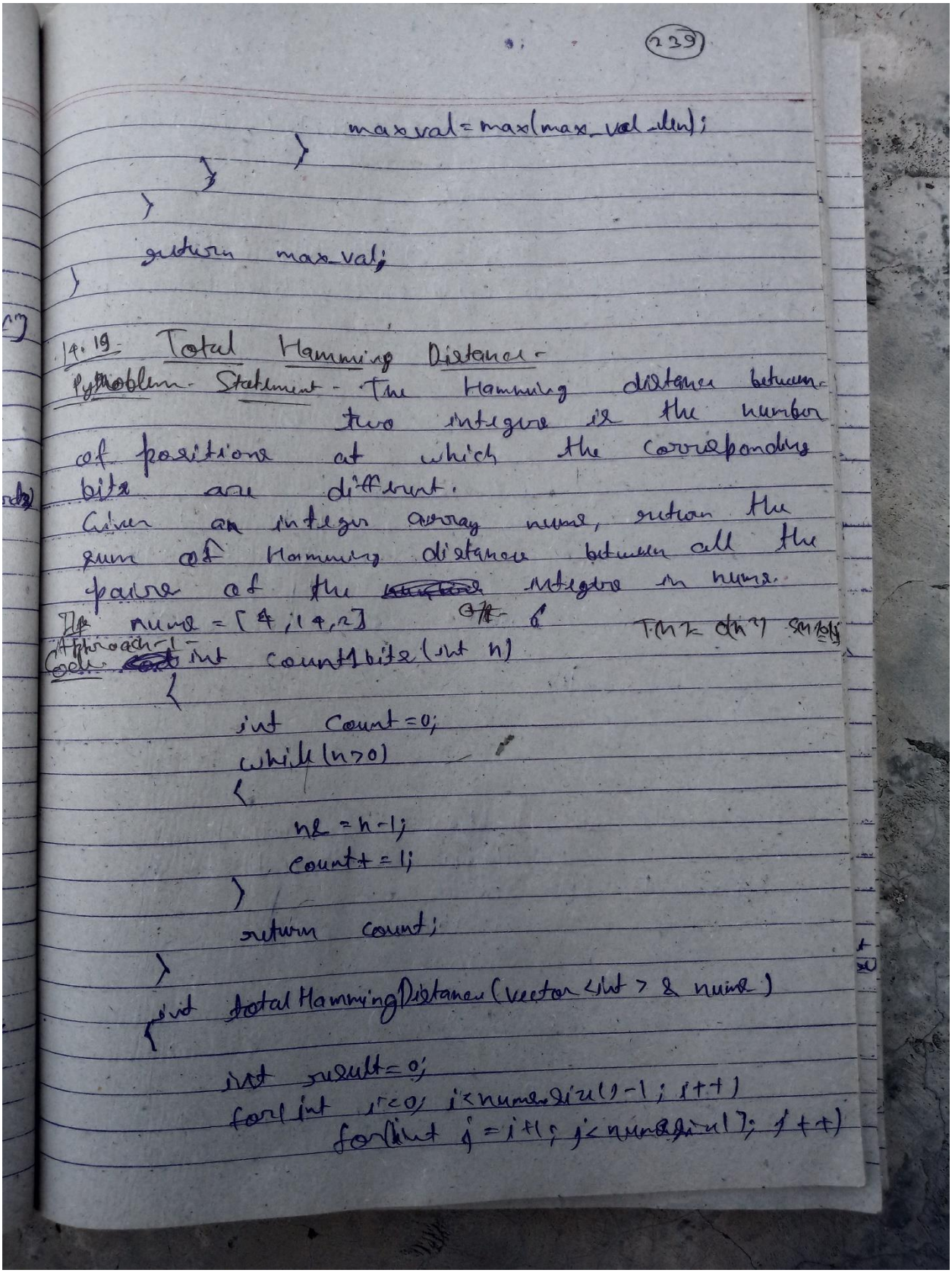


238

result[i] = result[i & (i+1)] + 1;
return result;
}

1814 Maximum Product of word Lengths -

Ex ["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]
Op 16 -
Explanation - The two words can be "abcw", "xtfn".
Approach 1 - TC = O(n² × len) SC = O(1)
Code - bool no_common_letters(string word1, string word2)
{
for(char c: word1)
if(word2.find(c) != string::npos)
return false;
return true;
}
int maxProduct(vector<string>& words)
{
int n = words.size();
int max_val = 0;
for(int i=0; i<n; i++)
{
for(int j=i+1; j<n; j++)
{
if(no_common_letters(words[i],
words[j]))
{
int len = words[i].length()
*words[j].length();



239

```
            maxval = max(max_val_dist);
        }
    }
    return max_val;
}
```

14.19 Total Hamming Distance -

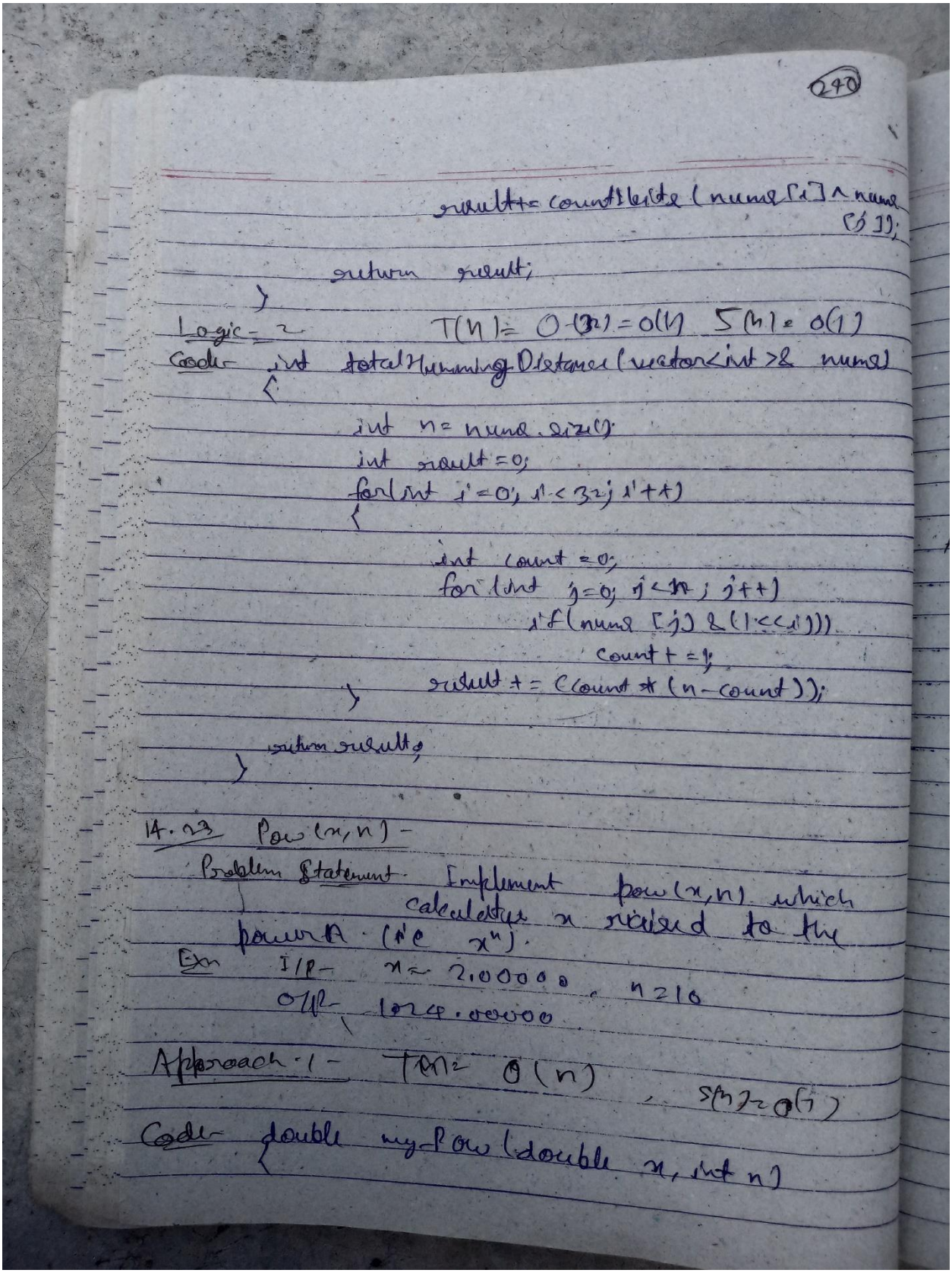
Problem Statement - The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given an integer array nums, return the sum of Hamming distances between all the pairs of the integers in nums.

Eg nums = [4, 14, 2]

Approach-1 -

```
int countBits(int n)
{
    int count = 0;
    while(n > 0)
    {
        n = n - 1;
        count++;
    }
    return count;
}
int totalHammingDistance(vector<int> & nums)
{
    int result = 0;
    for(int i = 0; i < nums.size() - 1; i++)
        for(int j = i + 1; j < nums.size(); j++)
```



240

result += countBits(nums[i] ^ nums[j]);

return result;

}

Logic - 2 T(n) = O(32) = O(1) S(n) = O(1)

Code - int totalHammingDistance(vector<int>& nums)

{

int n = nums.size();

int result = 0;

for(int i = 0; i < 32; i++)

{

int count = 0;

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

if(nums[j] & (1 << i))

count++;

result += (count * (n - count));

}

return result;

}

14.23 Pow(x, n) -

Problem Statement - Implement pow(x, n) which calculates x raised to the power n (i.e x^n).

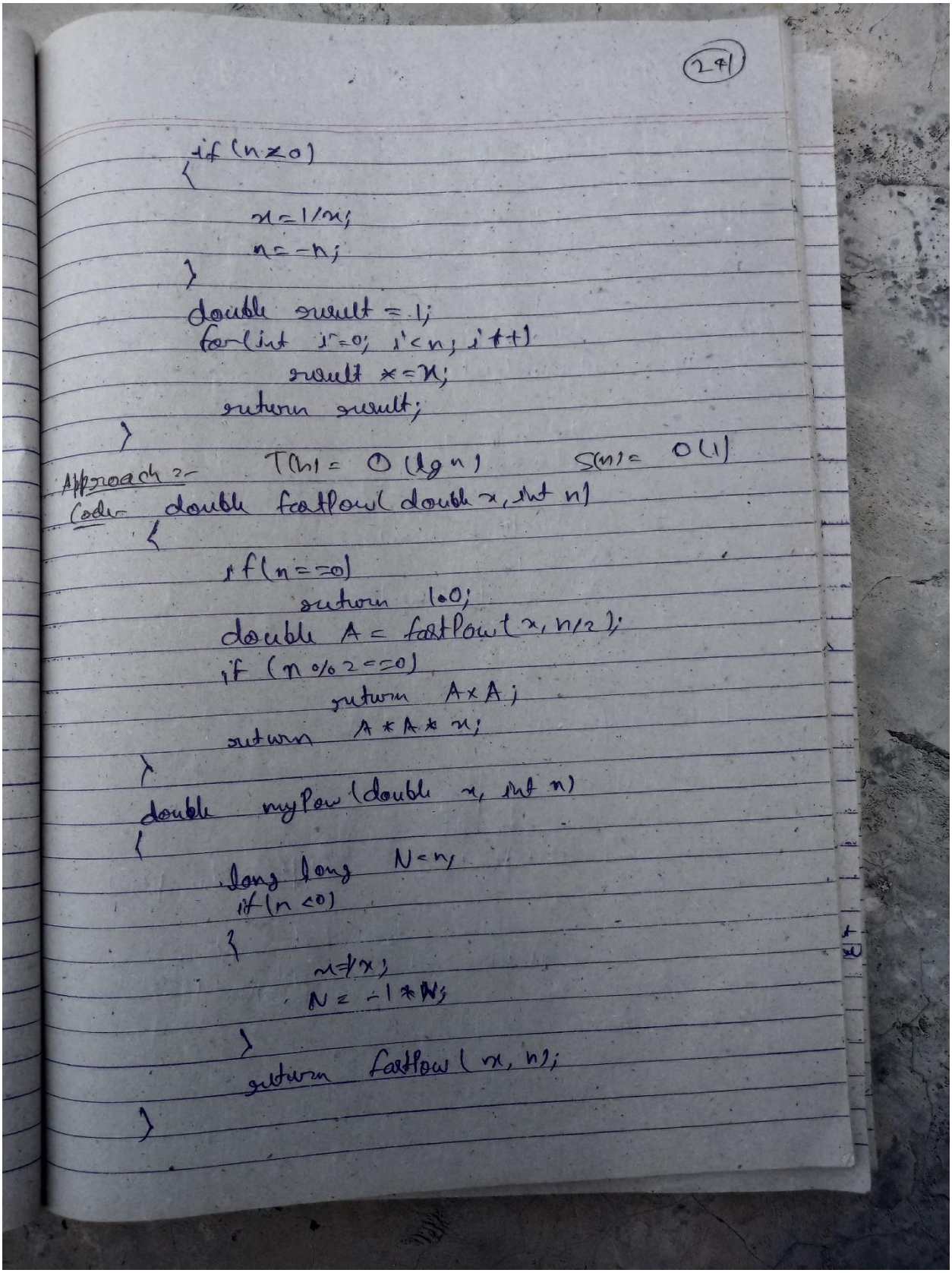
Ex I/P - x = 2.00000, n = 10

O/P - 1024.00000

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

Code - double myPow(double x, int n)

{



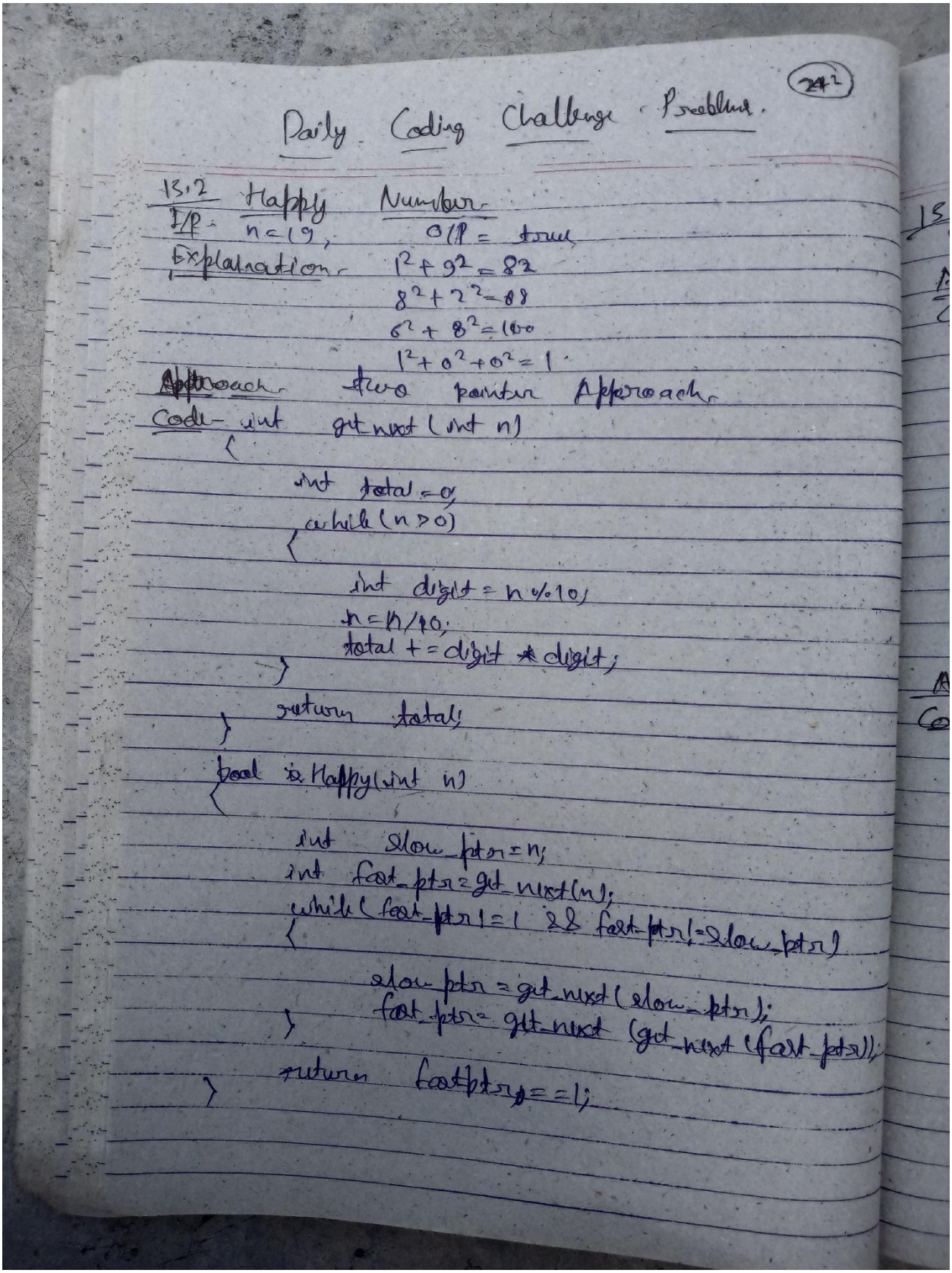
(27)

```
if (n<0)
{
    x = 1/x;
    n = -n;
}
double result = 1;
for(int i=0; i<n; i++)
    result *= x;
return result;
}
```

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

Code-

```
double fastPow(double x, int n)
{
    if(n==0)
        return 1.0;
    double A = fastPow(x, n/2);
    if (n%2==0)
        return A*A;
    return A*A*x;
}
double myPow(double x, int n)
{
    long long N=n;
    if(n<0)
    {
        x=1/x;
        N = -1*N;
    }
    return fastPow(x, N);
}
```



Daily Coding Challenge Problem. (242)

13.2 Happy Number

I/P: n=19, O/P = true

Explanation: $1^2+9^2=82$

$8^2+2^2=68$

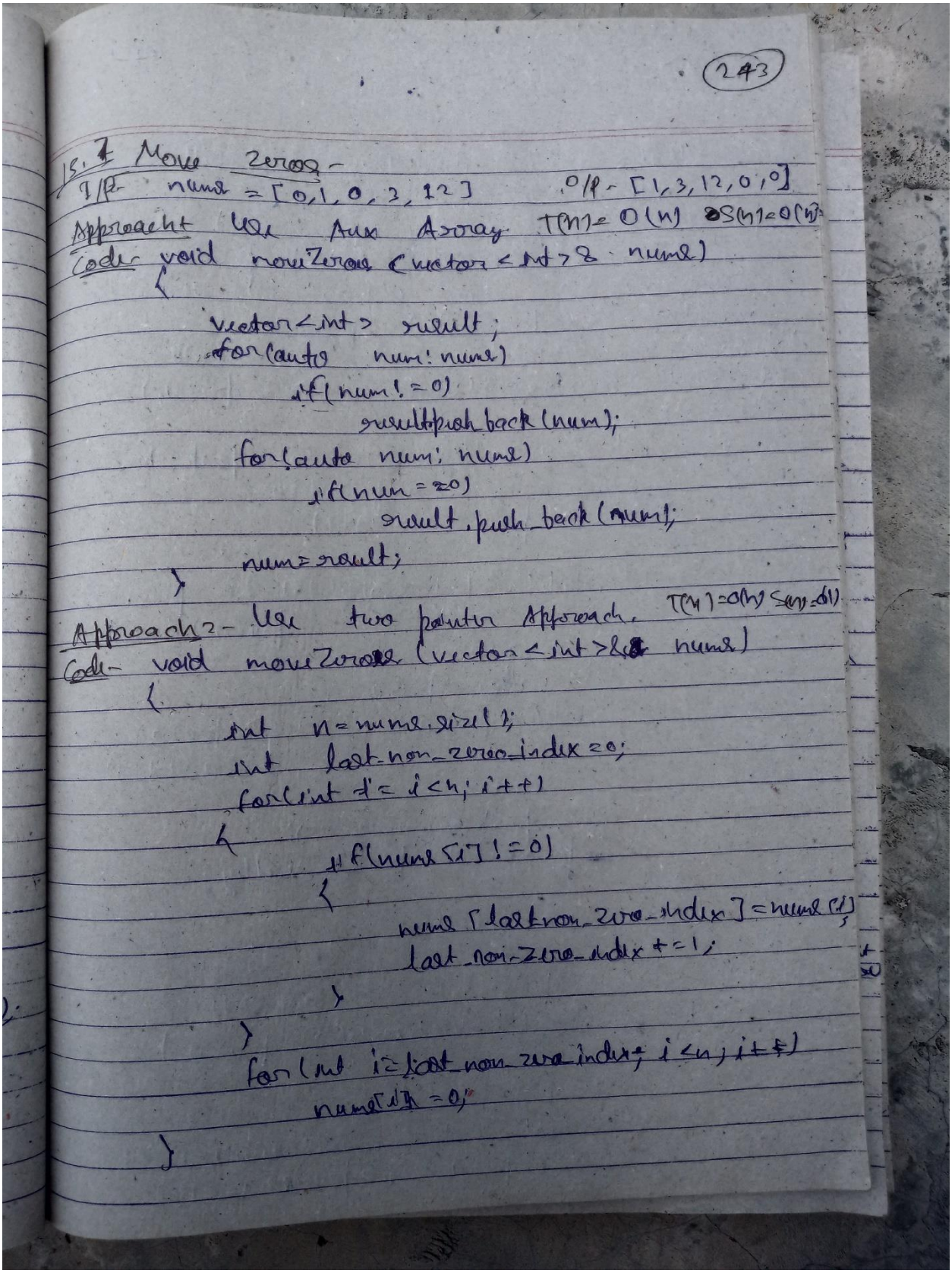
$6^2+8^2=100$

$1^2+0^2+0^2=1$

Approach: two pointer Approach.

Code:

```
int getnext(int n)
{
    int total=0;
    while(n>0)
    {
        int digit=n%10;
        n=n/10;
        total+=digit*digit;
    }
    return total;
}
bool isHappy(int n)
{
    int slow_ptr=n;
    int fast_ptr=get_next(n);
    while(fast_ptr!=1 && fast_ptr!=slow_ptr)
    {
        slow_ptr=get_next(slow_ptr);
        fast_ptr=get_next(get_next(fast_ptr));
    }
    return fastptr==1;
}
```



243

Q. 7 Move Zeroes -

I/P - nums = [0, 1, 0, 3, 12] O/P - [1, 3, 12, 0, 0]

Approach1 Use Aux Array T(n) = O(n) S(n) = O(n)

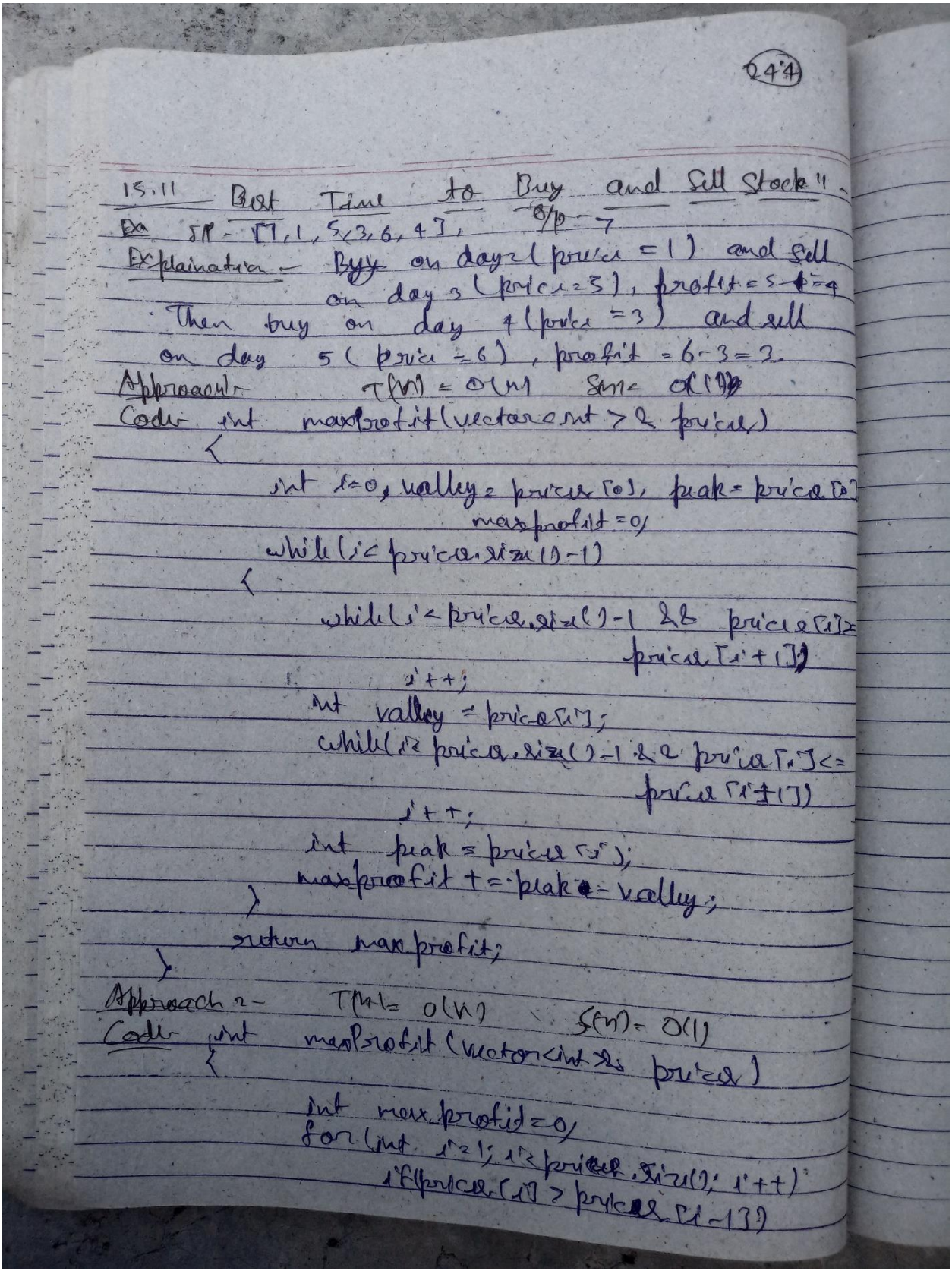
Code -

```
void moveZeroes(vector<int>& nums)
{
    vector<int> result;
    for(auto num: nums)
        if(num != 0)
            result.push_back(num);
    for(auto num: nums)
        if(num == 0)
            result.push_back(num);
    nums = result;
}
```

Approach2 - Use two pointer Approach. T(n) = O(n) S(n) = O(1)

Code -

```
void moveZeroes(vector<int>& nums)
{
    int n = nums.size();
    int last_non_zero_index = 0;
    for(int i = 0; i < n; i++)
    {
        if(nums[i] != 0)
        {
            nums[last_non_zero_index] = nums[i];
            last_non_zero_index += 1;
        }
    }
    for(int i = last_non_zero_index; i < n; i++)
        nums[i] = 0;
}
```



244

15.11 Best Time to Buy and Sell Stock" -

Ex- I/P - [7, 1, 5, 3, 6, 4], O/p - 7

Explanation - Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4
Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

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

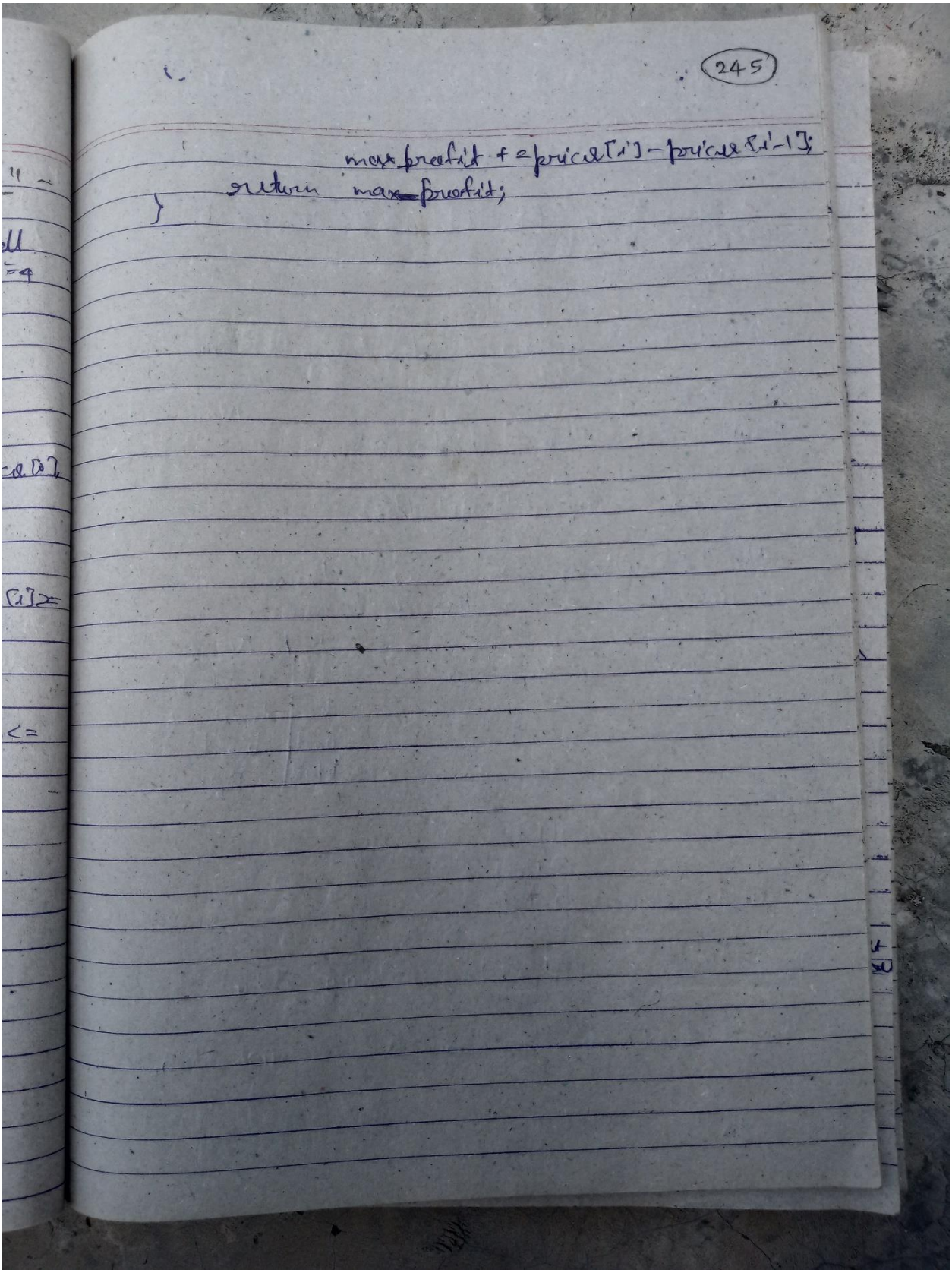
Code -

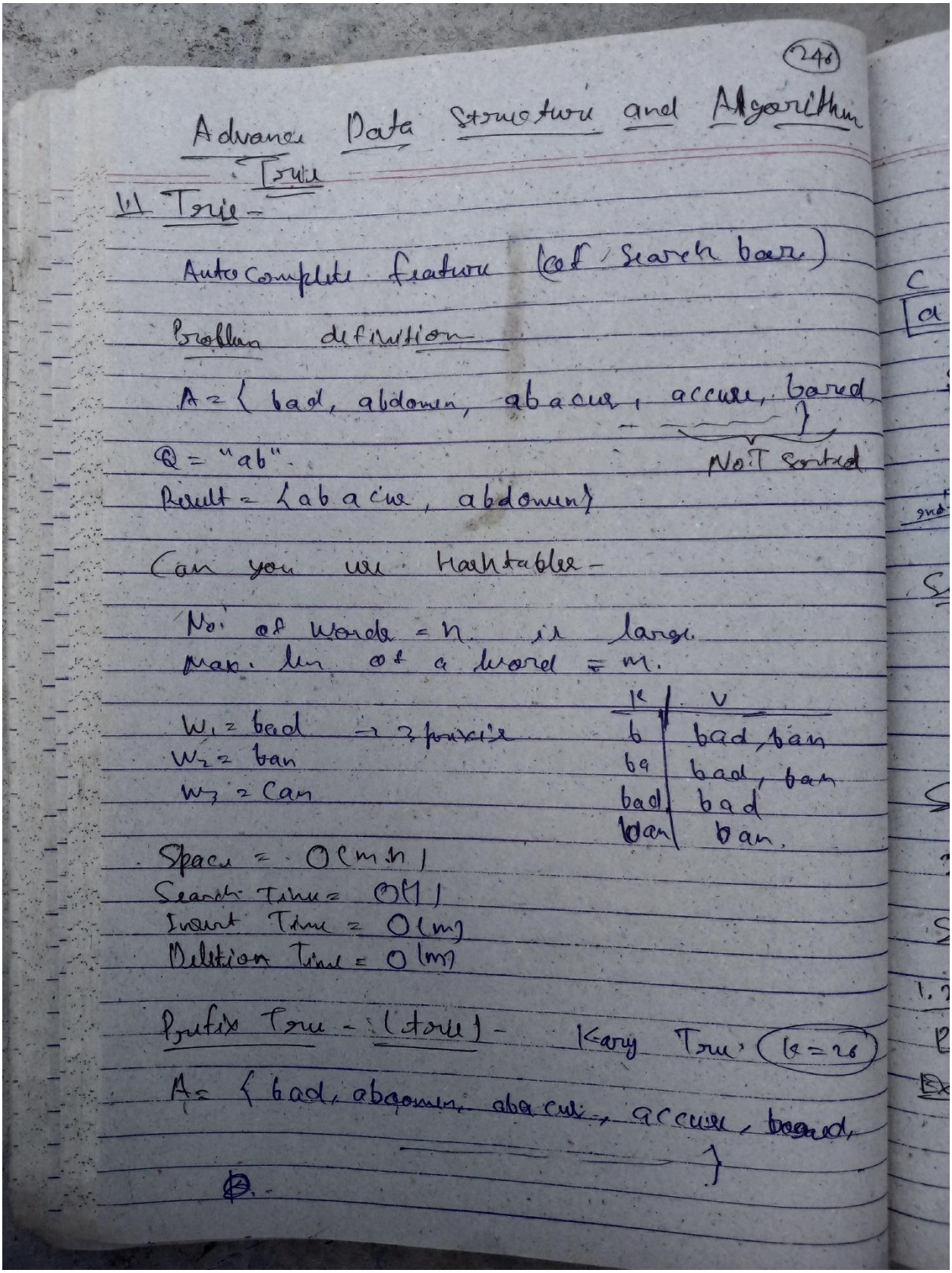
```
int maxProfit(vector<int>& prices)
{
    int i=0, valley = prices[0], peak = prices[0]
                                maxprofit = 0;
    while(i<prices.size()-1)
    {
        while(i<prices.size()-1 && prices[i]>=
                                prices[i+1])
            i++;
        int valley = prices[i];
        while(i<prices.size()-1 && prices[i]<=
                                prices[i+1])
            i++;
        int peak = prices[i];
        maxprofit += peak - valley;
    }
    return maxprofit;
}
```

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

Code -

```
int maxProfit(vector<int>& prices)
{
    int max_profit = 0;
    for(int i=1; i<prices.size(); i++)
        if(prices[i] > prices[i-1])
```


245
maxprofit += price[i] - price[i-1];
return maxprofit;
}



(246)

Advance Data Structure and Algorithm

Trie

1) Trie -

Autocomplete feature (of search bar)

Problem definition -

A = { bad, abdomen, abacus, accuse, board }

Not sorted

Q = "ab".

Result = {abacus, abdomen}

Can you use Hashtables -

No. of words = n. is large.

Max. len of a word = m.

k	v
b	bad, ban
ba	bad, ban
bad	bad
ban	ban.

w_1 = bad → 3 prefix

w_2 = ban

w_3 = can

Space = O(mn)

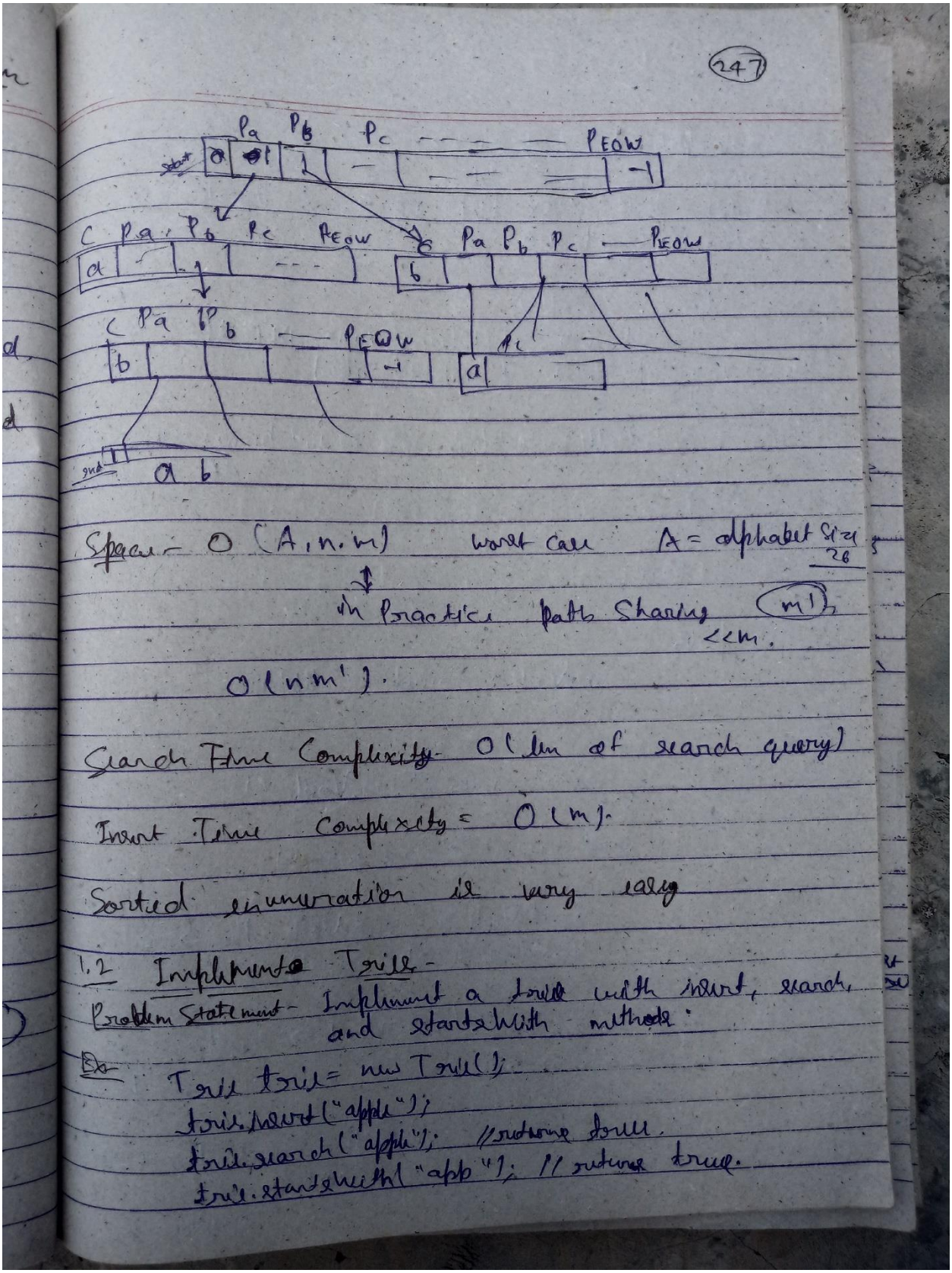
Search Time = O(1)

Insert Time = O(m)

Deletion Time = O(m)

Prefix Tree - (Trie) - K-ary Tree, (k = 26)

A = { bad, abdomen, abacus, accuse, board }



Space - O (A, n, m) worst case A = alphabet size 26

↕

In Practice path Sharing (m'), << m.

O(nm').

Search Time Complexity - O(len of search query)

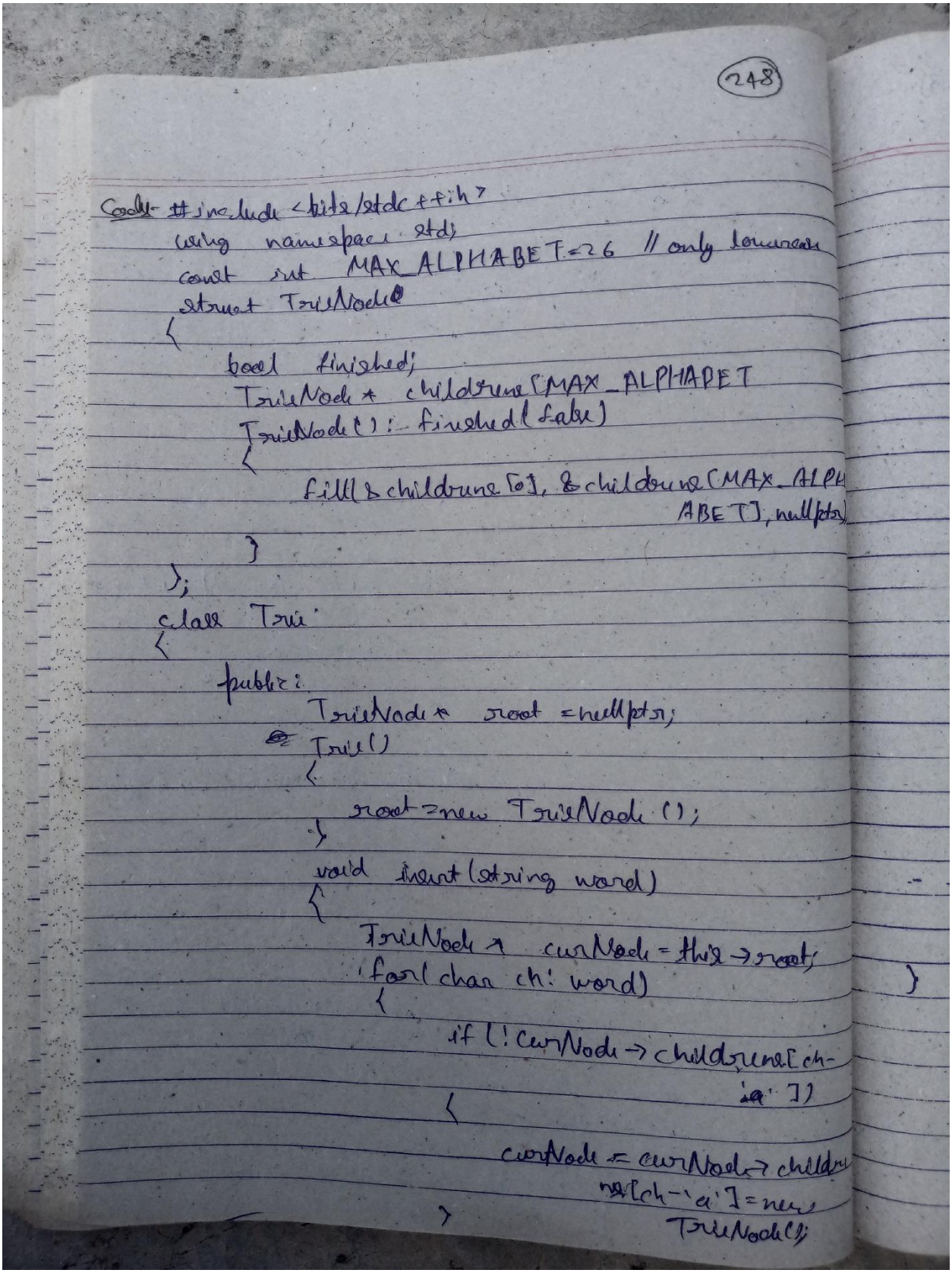
Insert Time Complexity = O(m).

Sorted enumeration is very easy

1.2 Implement a Trie -

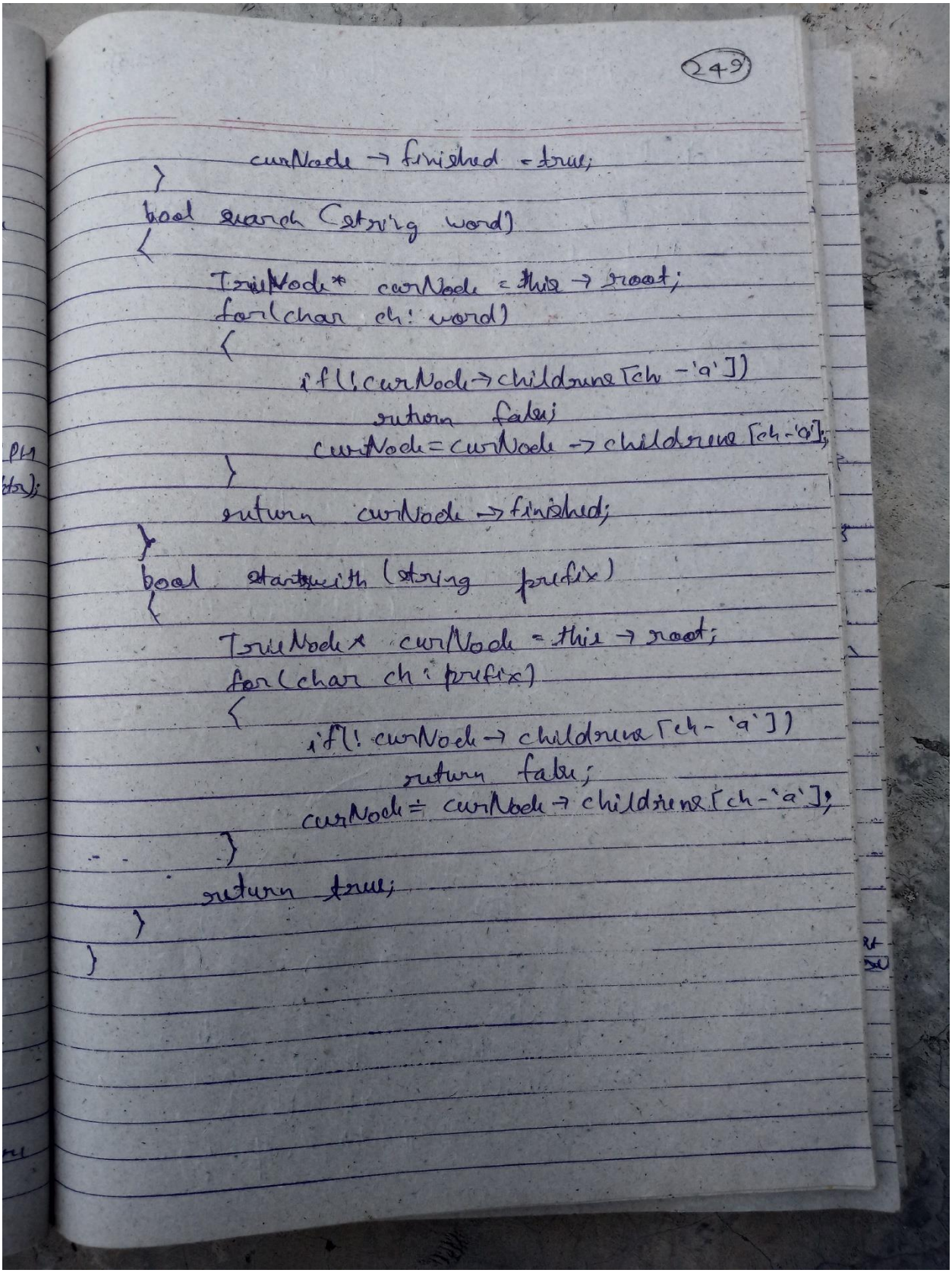
Problem Statement - Implement a trie with insert, search, and startsWith methods.

Ex- Trie trie = new Trie();
trie.insert("apple");
trie.search("apple"); // returns true.
trie.startsWith("app"); // returns true.



248

```
Code #include <bits/stdc++.h>
using namespace std;
const int MAX_ALPHABET=26 // only lowercase
struct TrieNode
{
    bool finished;
    TrieNode * childrens[MAX_ALPHABET
    TrieNode(): finished(false)
    {
        fill(&childrens[0], &childrens[MAX_ALPH
                                    ABET], nullptr);
    }
};
class Trie
{
    public:
        TrieNode * root =nullptr;
        Trie()
        {
            root=new TrieNode();
        }
        void insert(string word)
        {
            TrieNode * curNode = this->root;
            for(char ch: word)
            {
                if(!curNode->childrens[ch-
                                    'a']
                {
                curNode = curNode->childr
                ens[ch-'a']=new
                    TrieNode();
```



```
        curNode -> finished = true;
    }
    bool search(string word)
    {
        TrieNode* curNode = this -> root;
        for(char ch: word)
        {
            if(!curNode->children[ch - 'a'])
                return false;
            curNode = curNode -> children[ch-'a'];
        }
        return curNode -> finished;
    }
    bool startsWith(string prefix)
    {
        TrieNode* curNode = this -> root;
        for(char ch: prefix)
        {
            if(!curNode -> children[ch - 'a'])
                return false;
            curNode = curNode -> children[ch-'a'];
        }
        return true;
    }
}
```

RED-BLACK Tree

250

Red Black Tree -

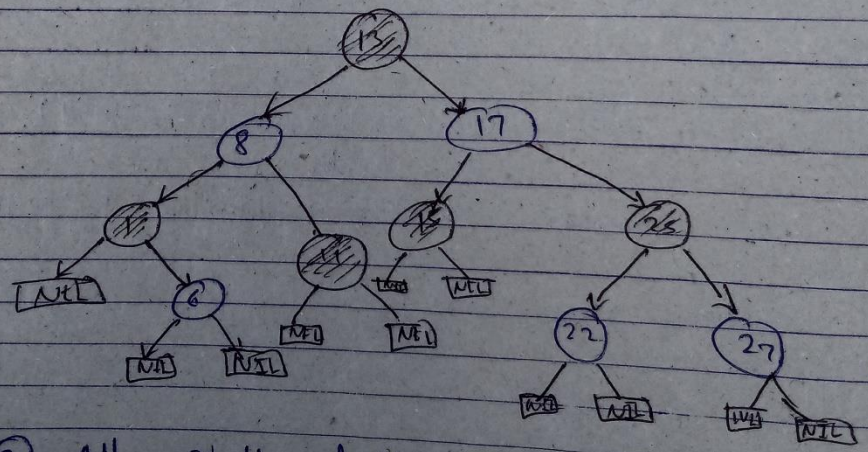
Self Balancing Tree h = O(log n)

Eg- AVL tree, B-Tree

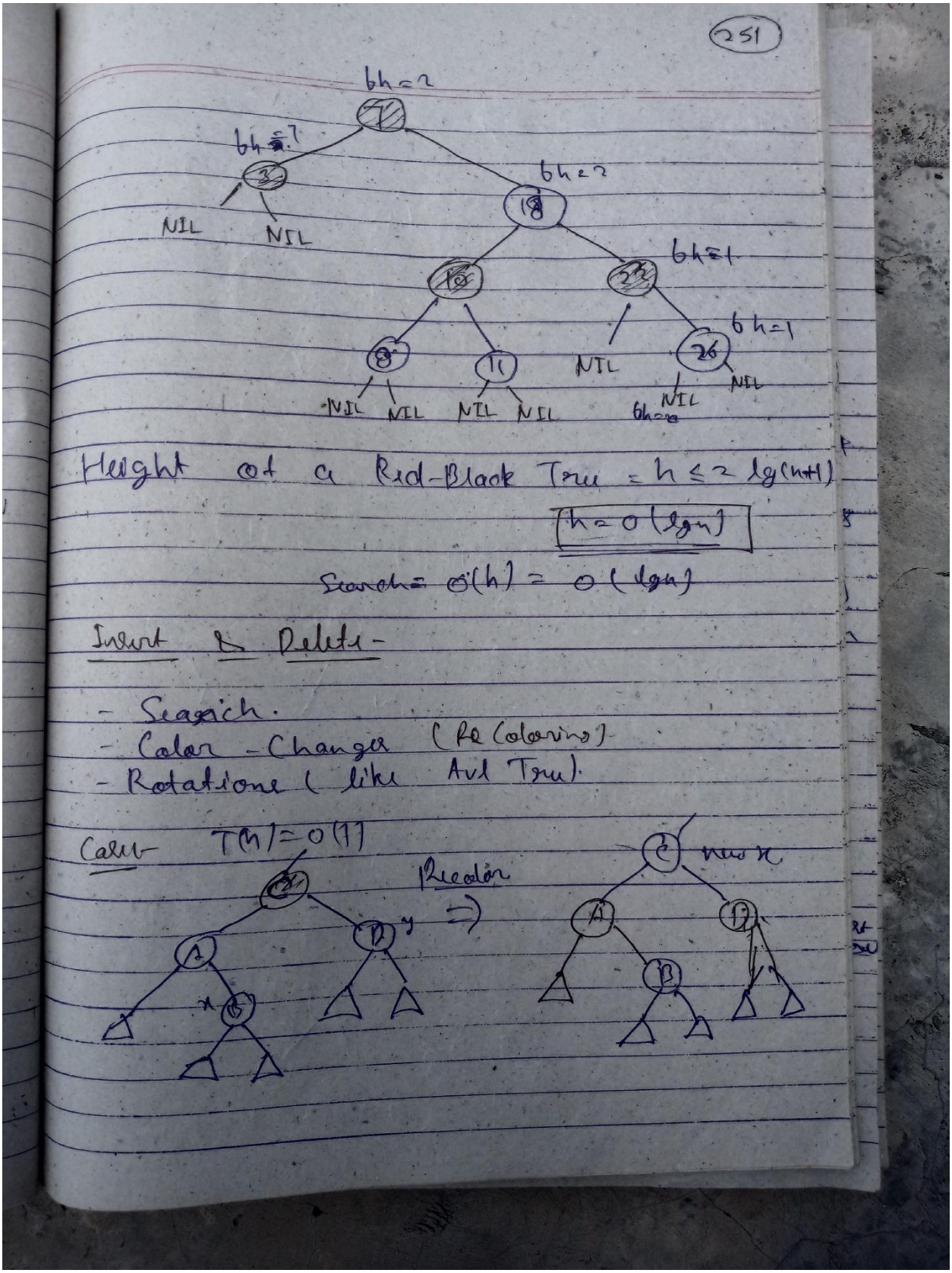
BST = Balancing BST

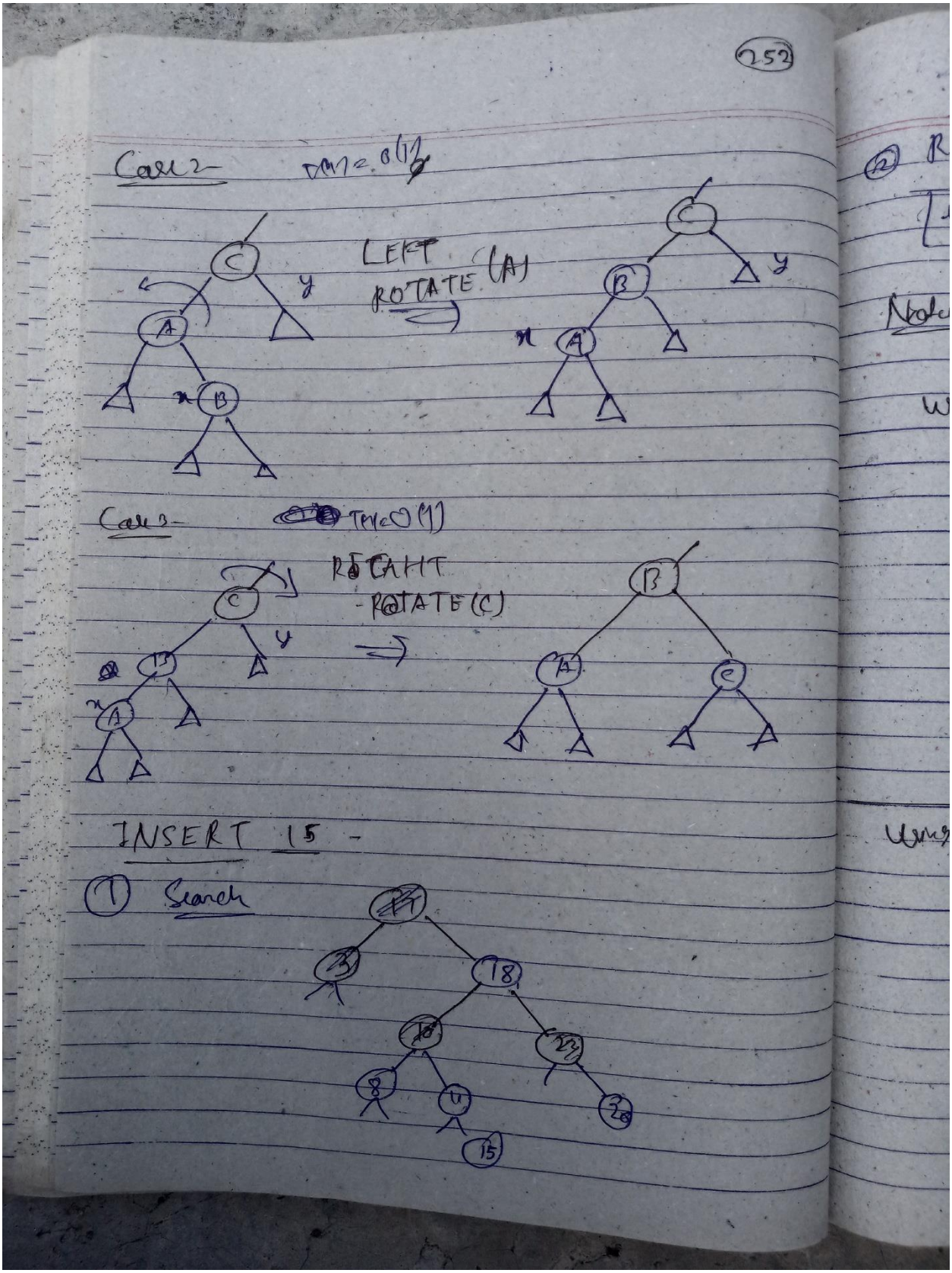
Definition - (RBT)

1. Node color Black / Red
2. Root & NIL (Leaves) are Black
3. A RED node has a black parent. (Red Black Property)



4. All simple paths from a node x to all descendant leaves have same number of black nodes (Black Height Property)


251
bh=2
bh=1
bh=2
NIL
NIL
bh=1
bh=1
NIL
NIL
NIL
NIL
NIL
NIL
NIL
Height of a Red-Black Tree = h ≤ 2 lg(n+1)
h = O(lg n)
Search = O(h) = O(lg n)
Insert & Delete -
- Search.
- Color - Changes (Re Coloring)
- Rotations (like Avl Tree).
Color - T(n) = O(1)
Recolor
new x


252
Case 2-
LEFT ROTATE (A)
Case 3-
RIGHT ROTATE (C)
INSERT 15 -
1) Search

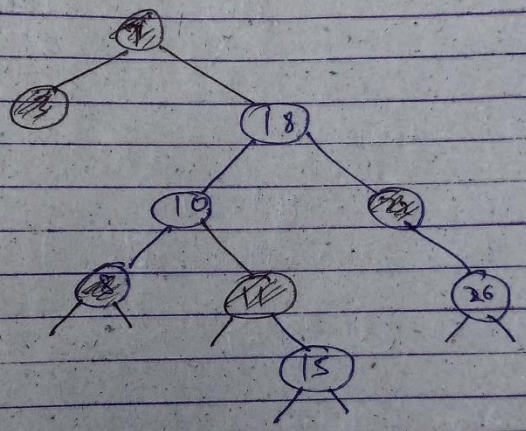
253

② Recolor

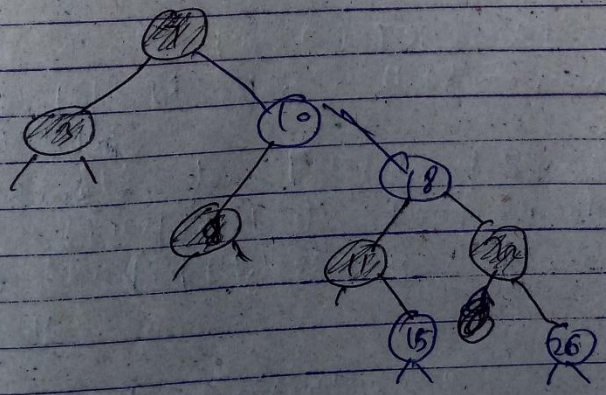
if (15) was a black node bh violation.

Note - Every new Node that inserts is by default RED color.

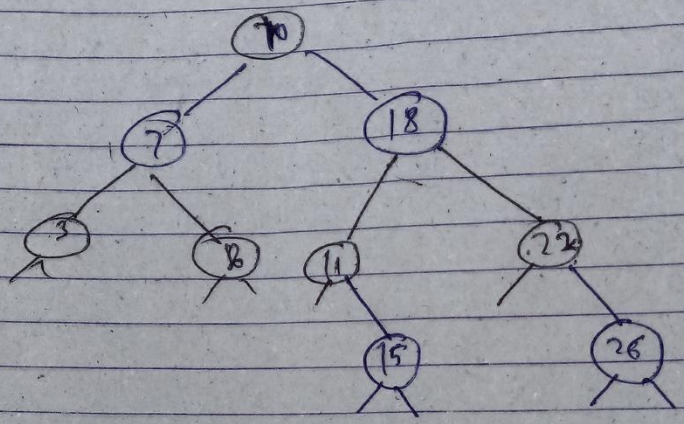
Using Call 1- Recolor.



Using Call 3- RR(18)



Using Case LR(7)



22

T(n) of insertion -

Search = O(lgn)

Fixing = O(1 . lgn)

T(n) = O(lgn)

Algo for Insertion -

RB-INSERT(T, x) -

TREE-INSERT(T, x)

color[x] ← RED only RB property 3 can be violated.

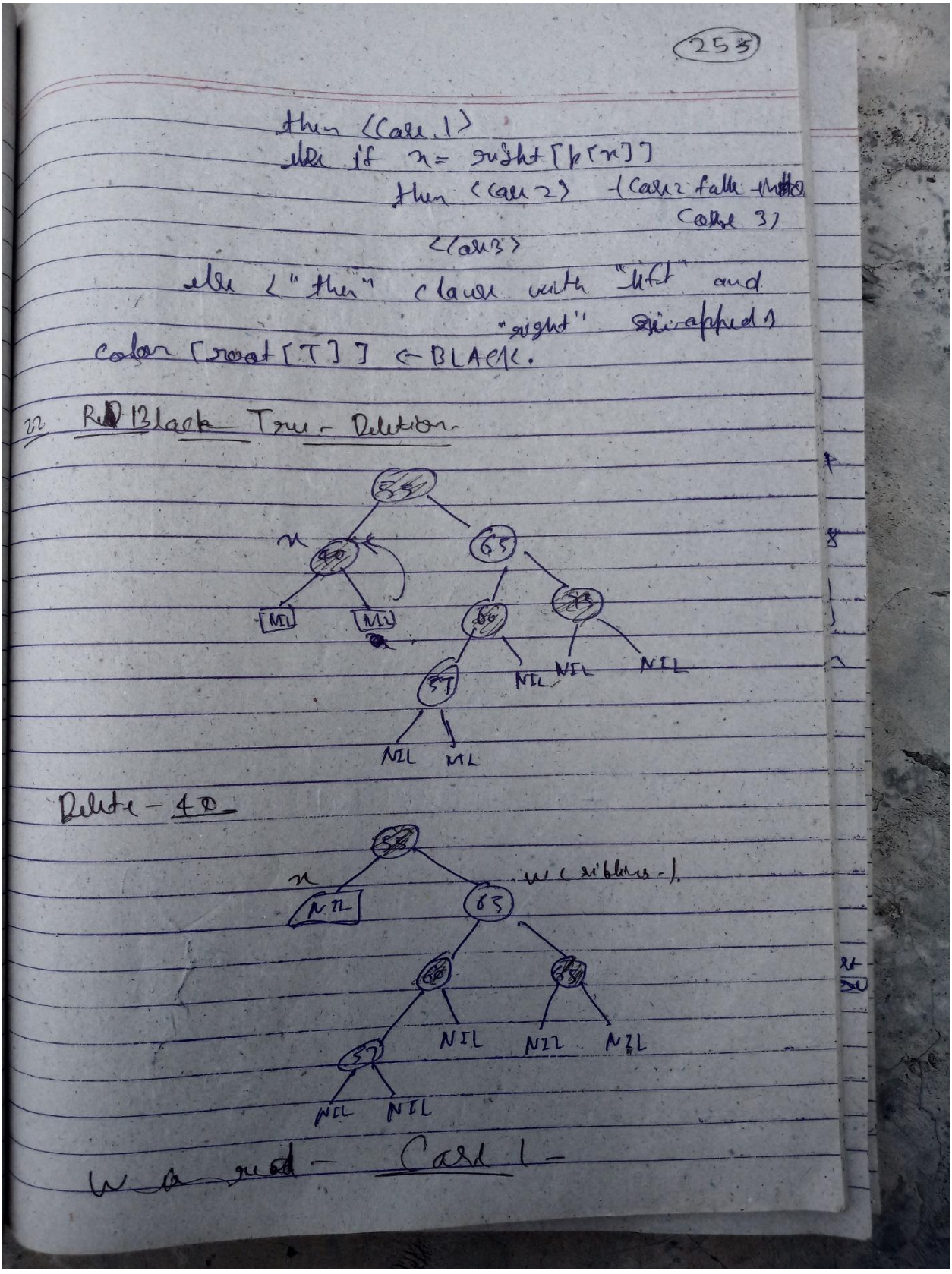
while x ≠ root[T] and color[p[x]] = RED

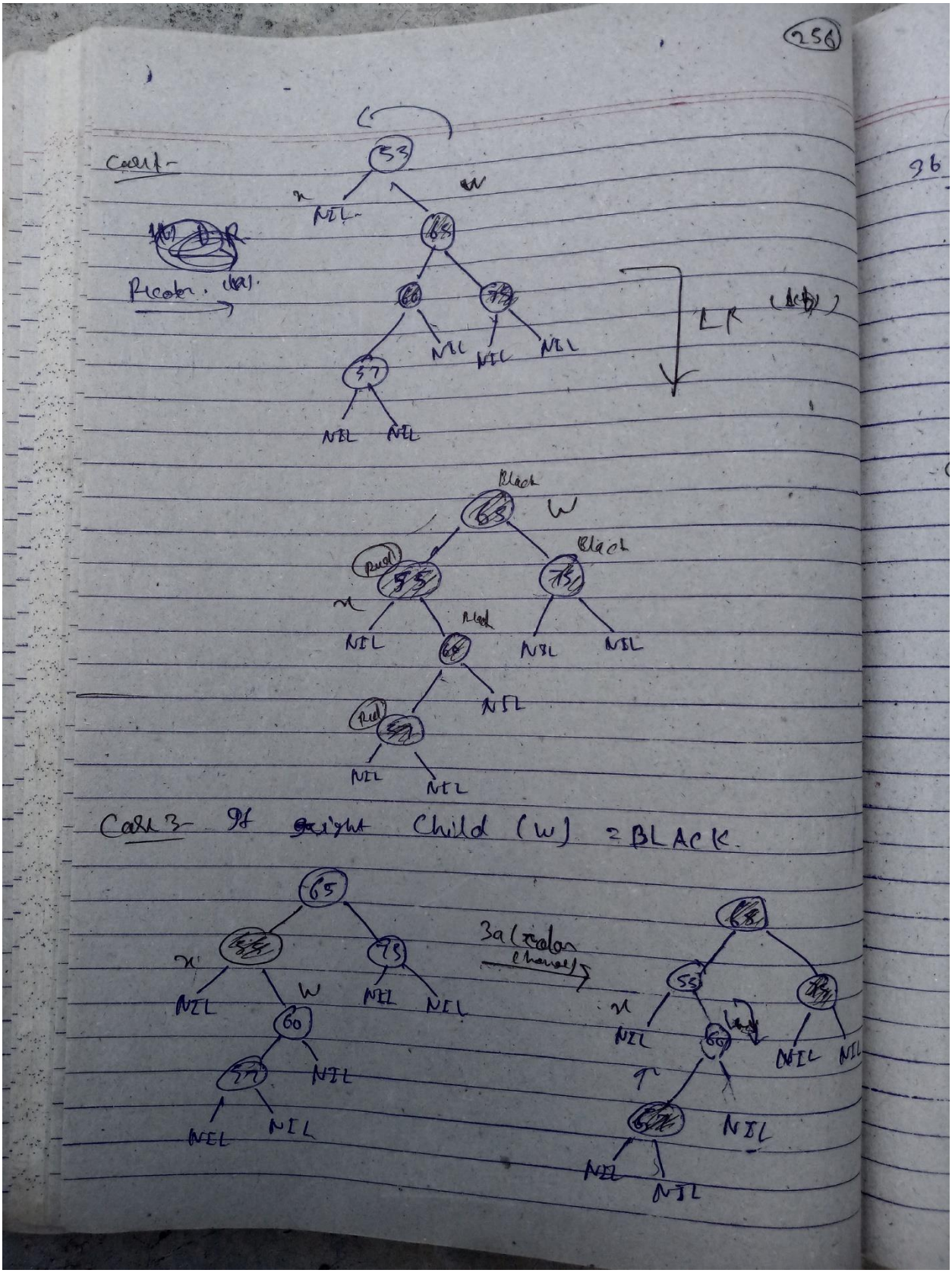
do if p[x] = left[p[p[x]]]

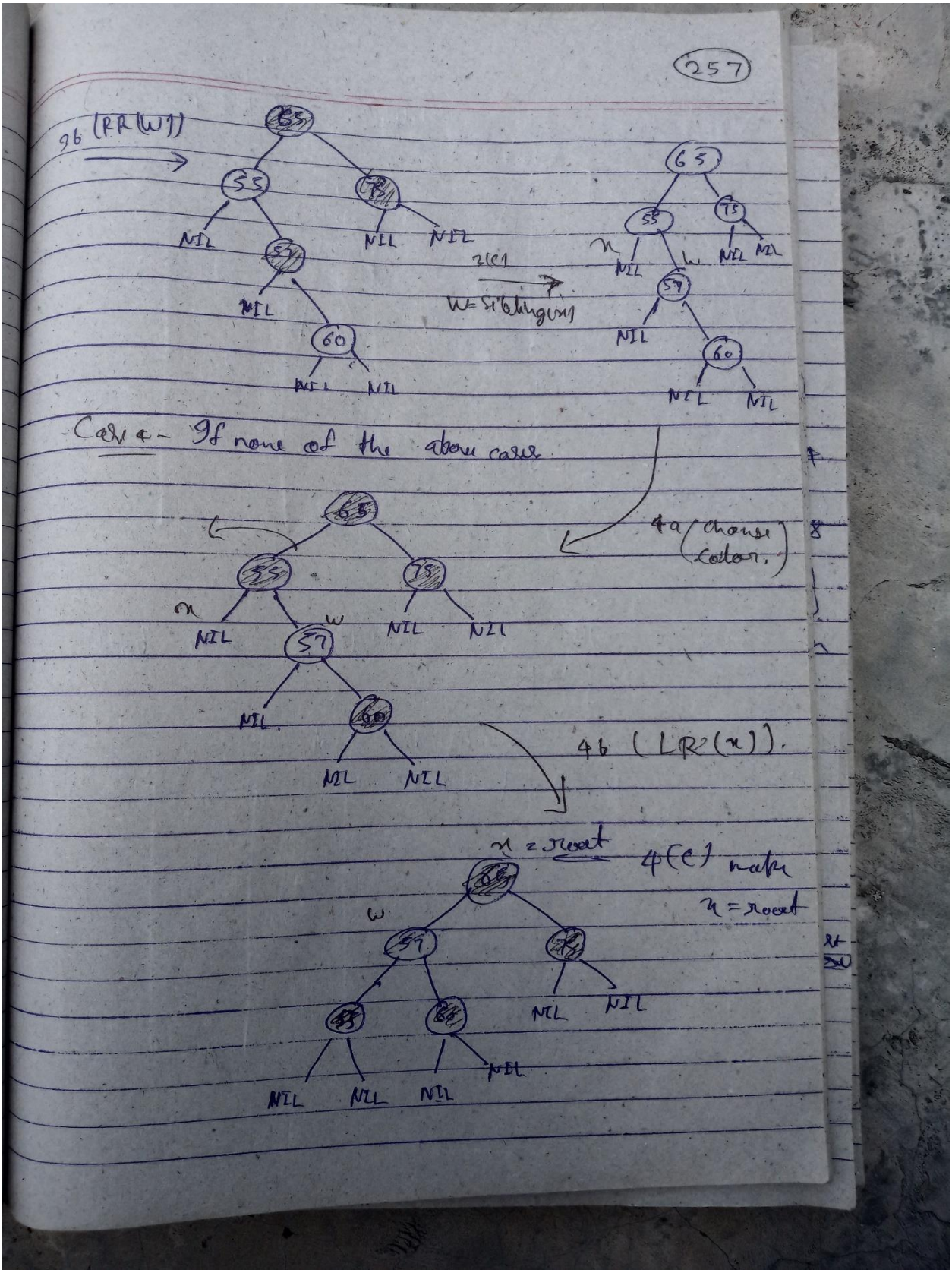
then y ← right[p[p[x]]]

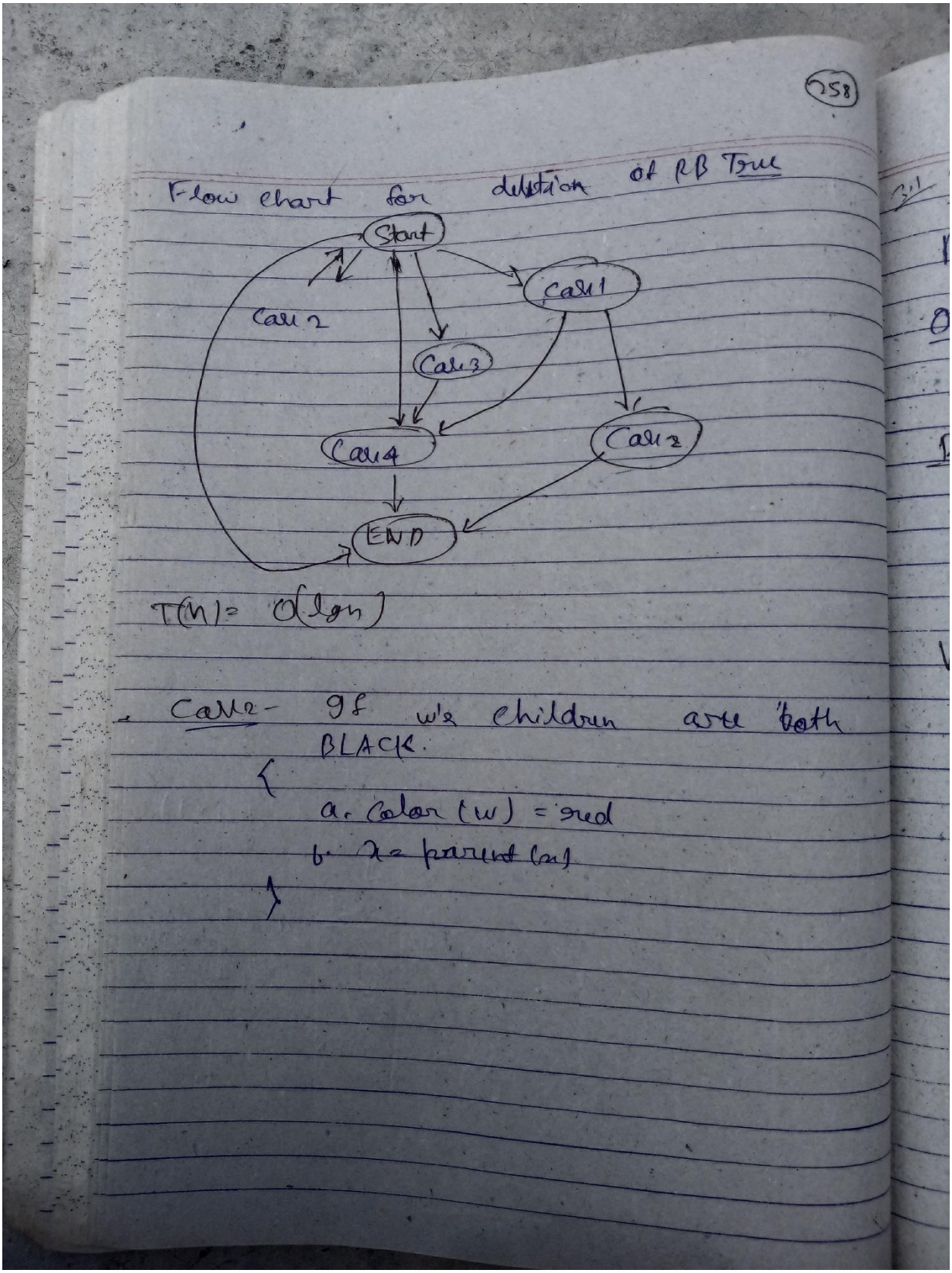
y = aunt/uncle of x

if color[y] = RED


253
then (Case 1)
else if x = right[p[x]]
then (case 2) - (Case 2 falls into Case 3)
(Case 3)
else ("then" clause with "left" and "right" applied)
color[root[T]] ← BLACK.
Red Black Tree - Deletion
x
NIL
NIL
NIL
NIL
NIL
NIL
NIL
Delete - 40
x
NIL
NIL
NIL
NIL
NIL
NIL
Worked - Case 1 -


256
Case 1-
Recolor. (RB).
LR (RB)
Black
Red
Case 3- If Right Child (w) = BLACK.
3a (recolor
Change)


257
3b (RR(W))
NIL
NIL
NIL
NIL
60
NIL
NIL
3(c)
W = Sibling(x)
65
55
75
NIL
NIL
57
NIL
60
NIL
NIL
Case 4 - If none of the above cases
4a (change color)
65
55
75
x
NIL
w
57
NIL
NIL
NIL
60
NIL
NIL
4b (LR(x)).
x = root
4(c) make
x = root
w
57
NIL
NIL
NIL
NIL
NIL
NIL


258
Flow chart for deletion of RB Tree
Start
Call 1
Call 2
Call 3
Call 4
Call 2
END
T(n) = O(log n)
Case 2 - If w's children are both BLACK.
{
a. Color (w) = red
b. x = parent (x)
}

B-Tree & B+ Tree

(259)

Files & Indexing-

DB - Tables → Disk as files (non volatile)

OS - access data on the disk fast
↳ search, read, write, modify, delete.

DSAlgo { Linked List, Array, Binary Search Tree / AVL - tree, Hash tables } data in RAM

DS - on disks / secondary memory.

Why do we need an index?

DS str. access based on that attribute is fast.

id: index

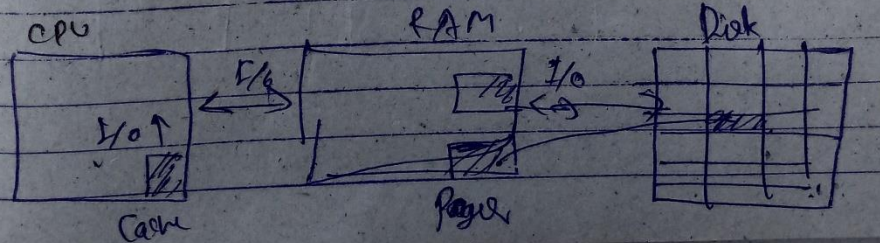
Search by name - prefix search?

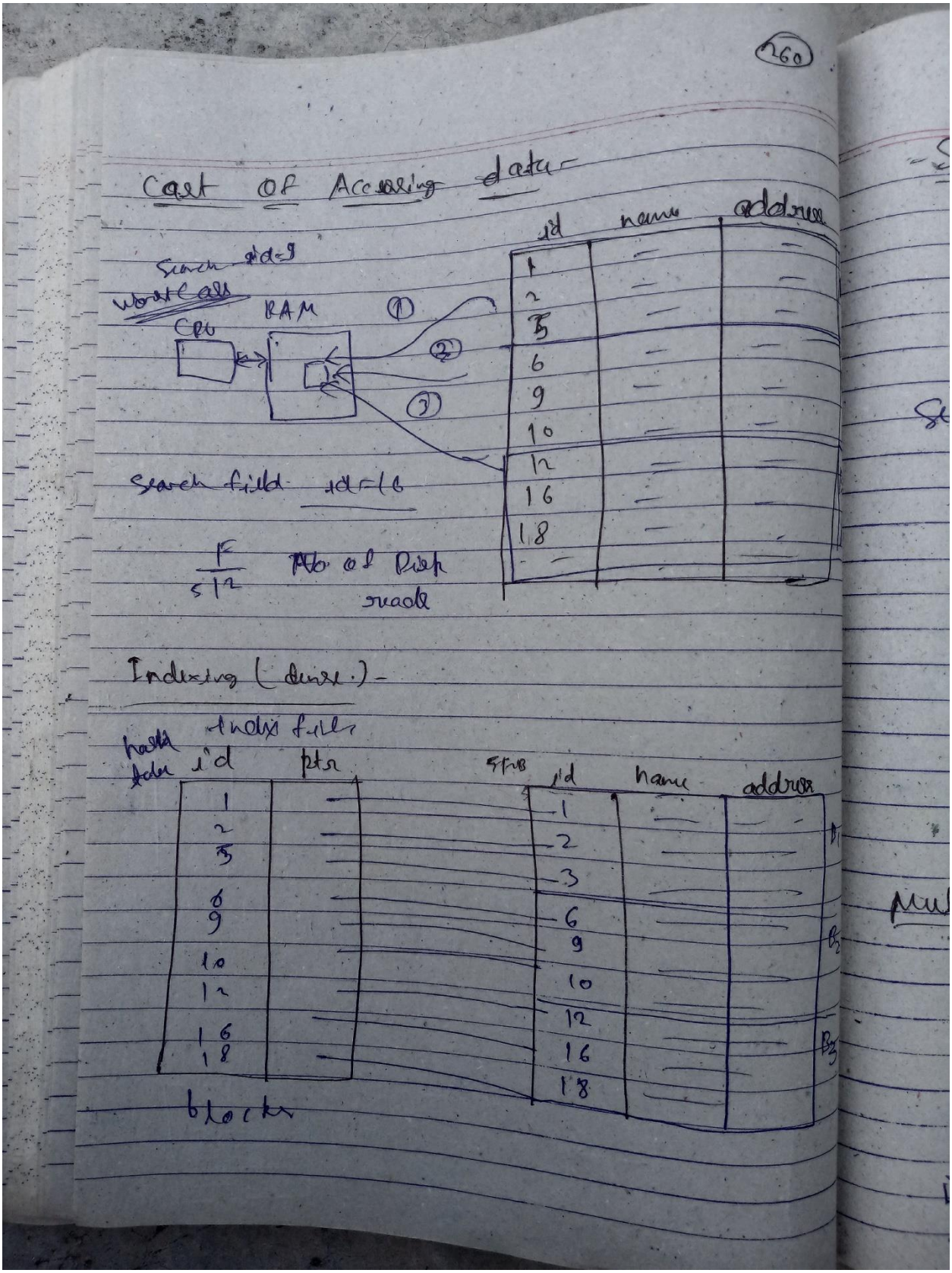
id	name	addr
1	n_1	a_1
4	n_2	a_2
6	n_3	a_3

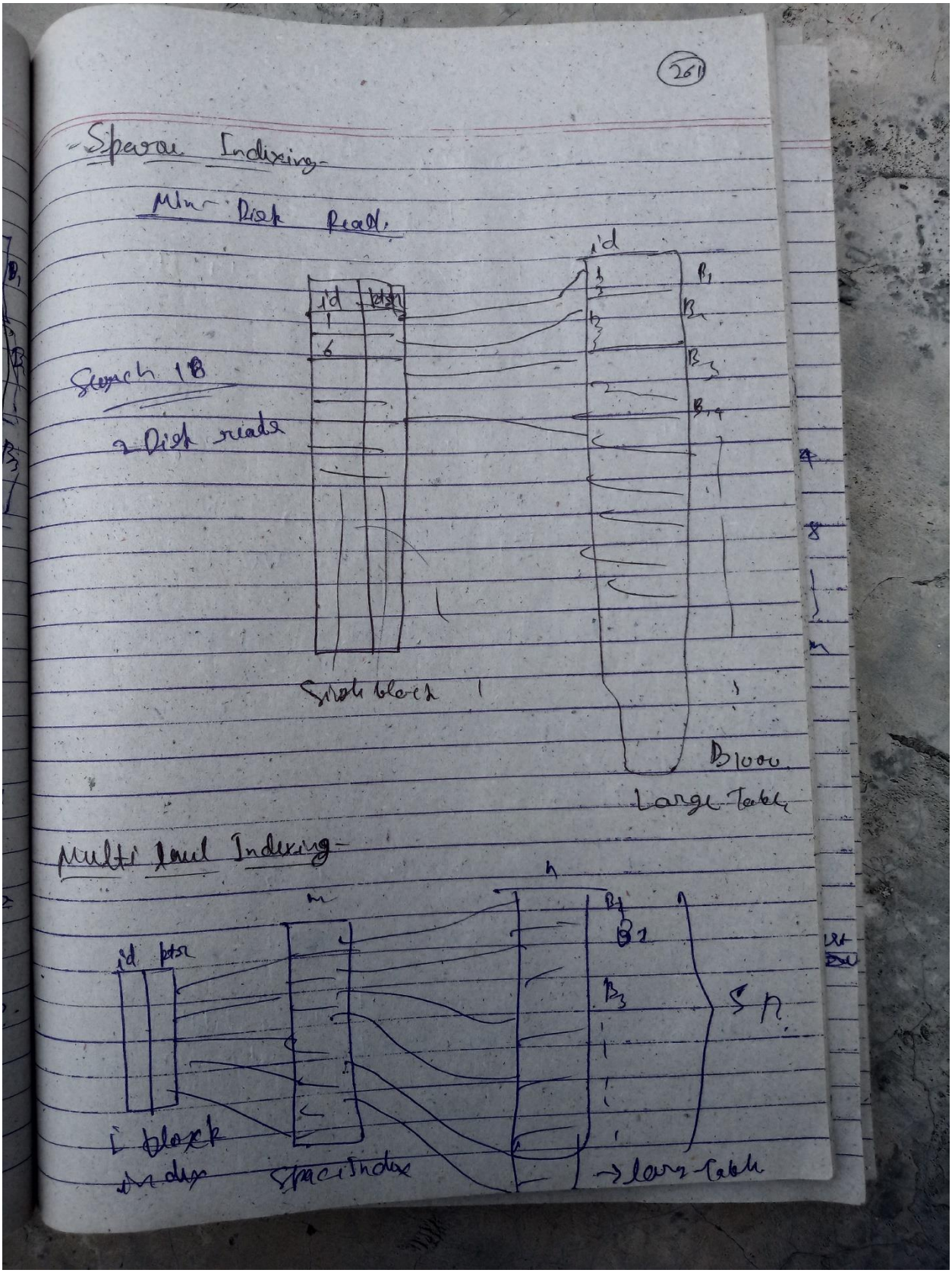
We can have multiple indices on a given table.

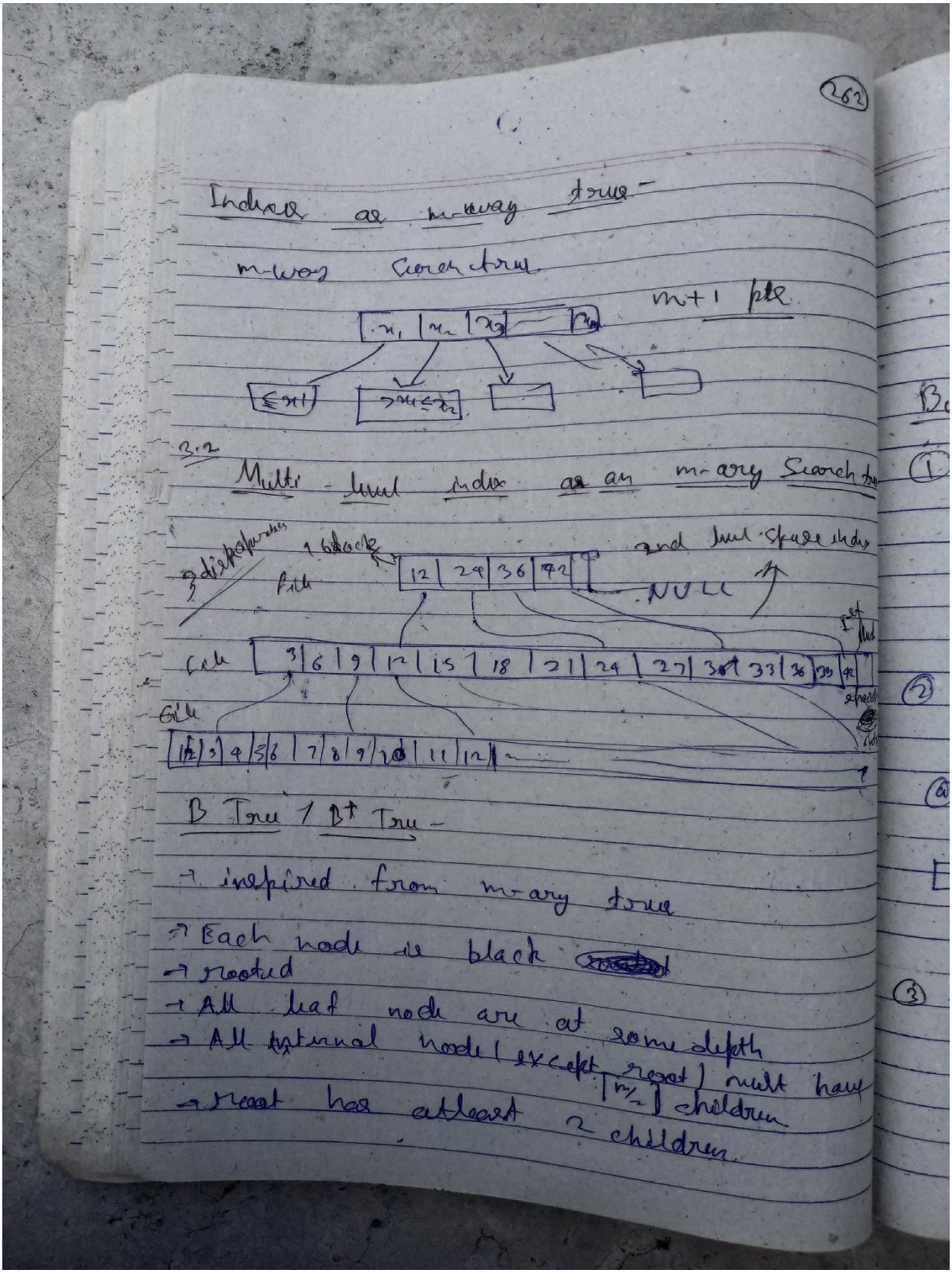
num str str.

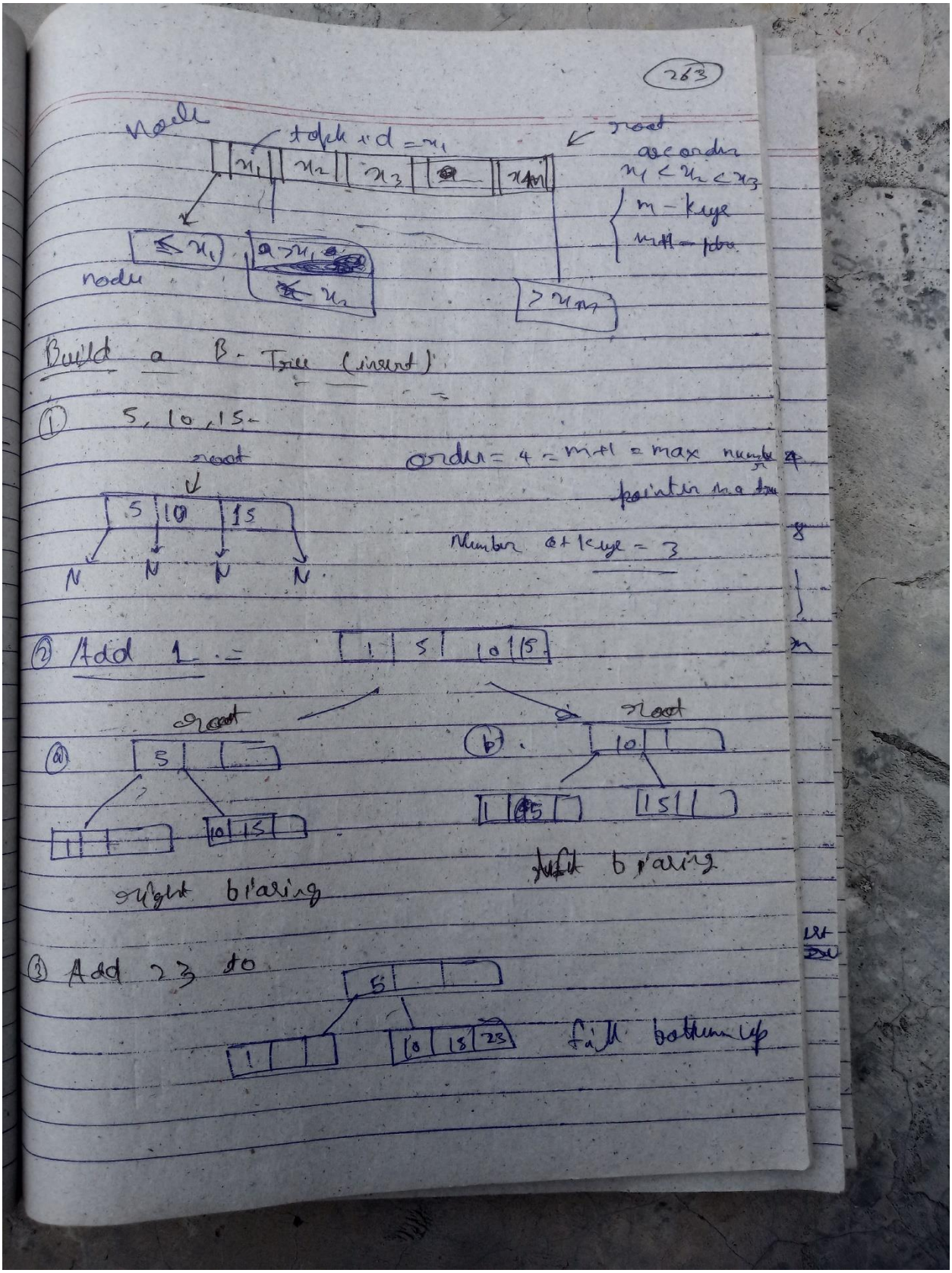
Accessing data on a disk

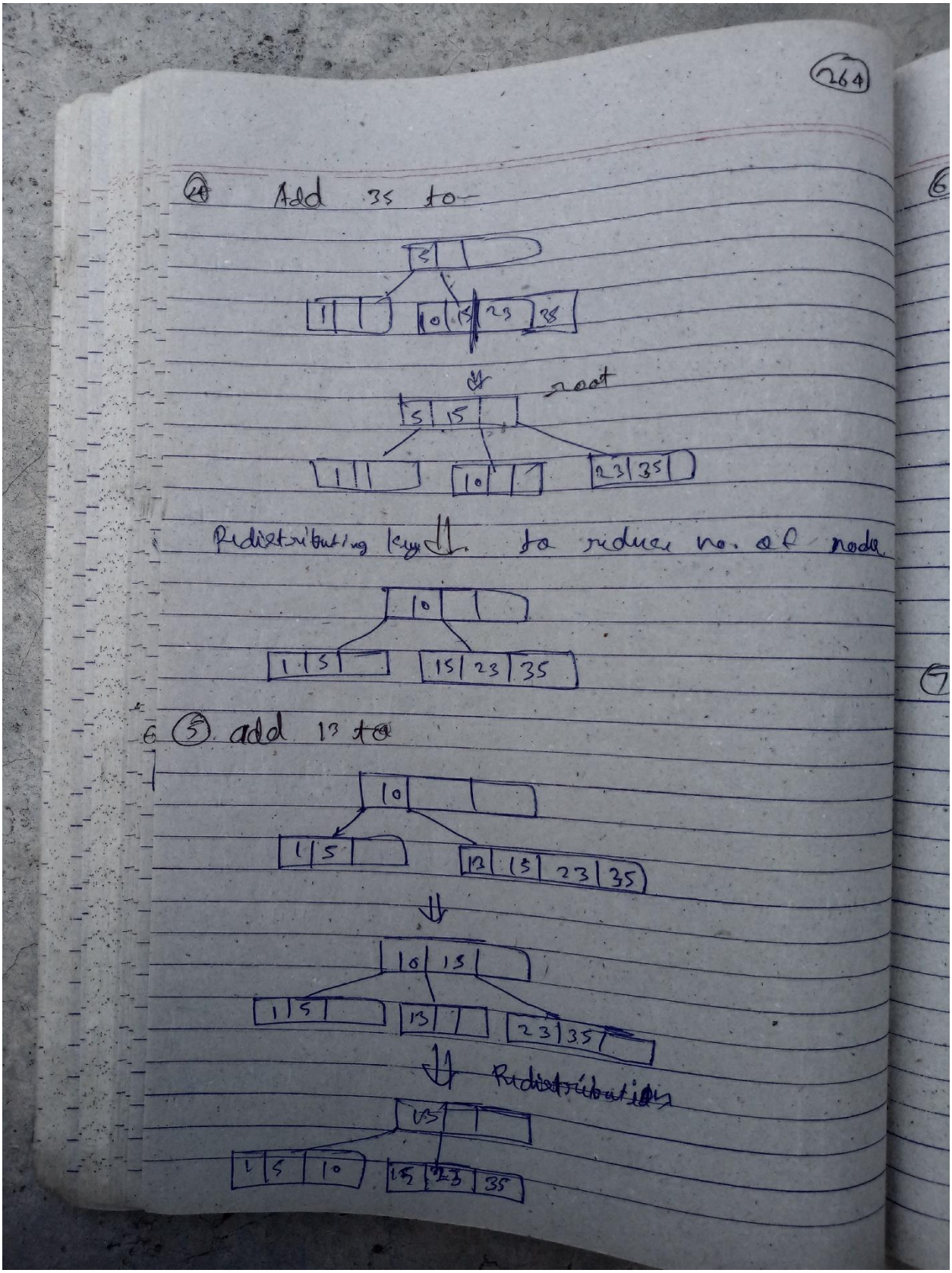


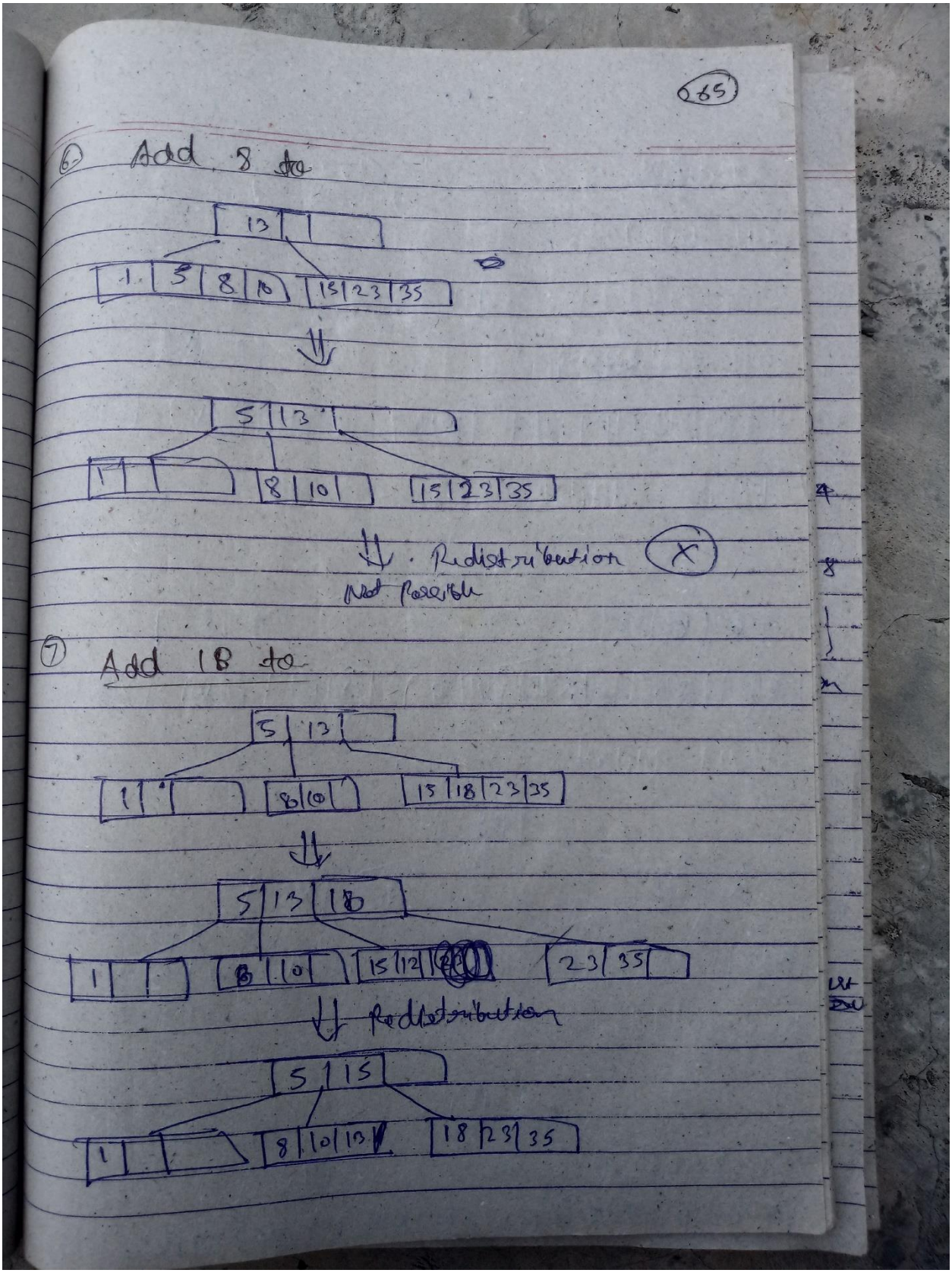

260
Cost of Accessing data
Search id=9
worst case
CPU
RAM
id
name
address
1
2
3
6
9
10
12
16
18
Search field id=16
5
12
No of Disk reads
Indexing (Sparse)-
hash table
index field
id
ptr
Disk
id
name
address
1
2
3
6
9
10
12
16
18
blocks
B1
B2
B3

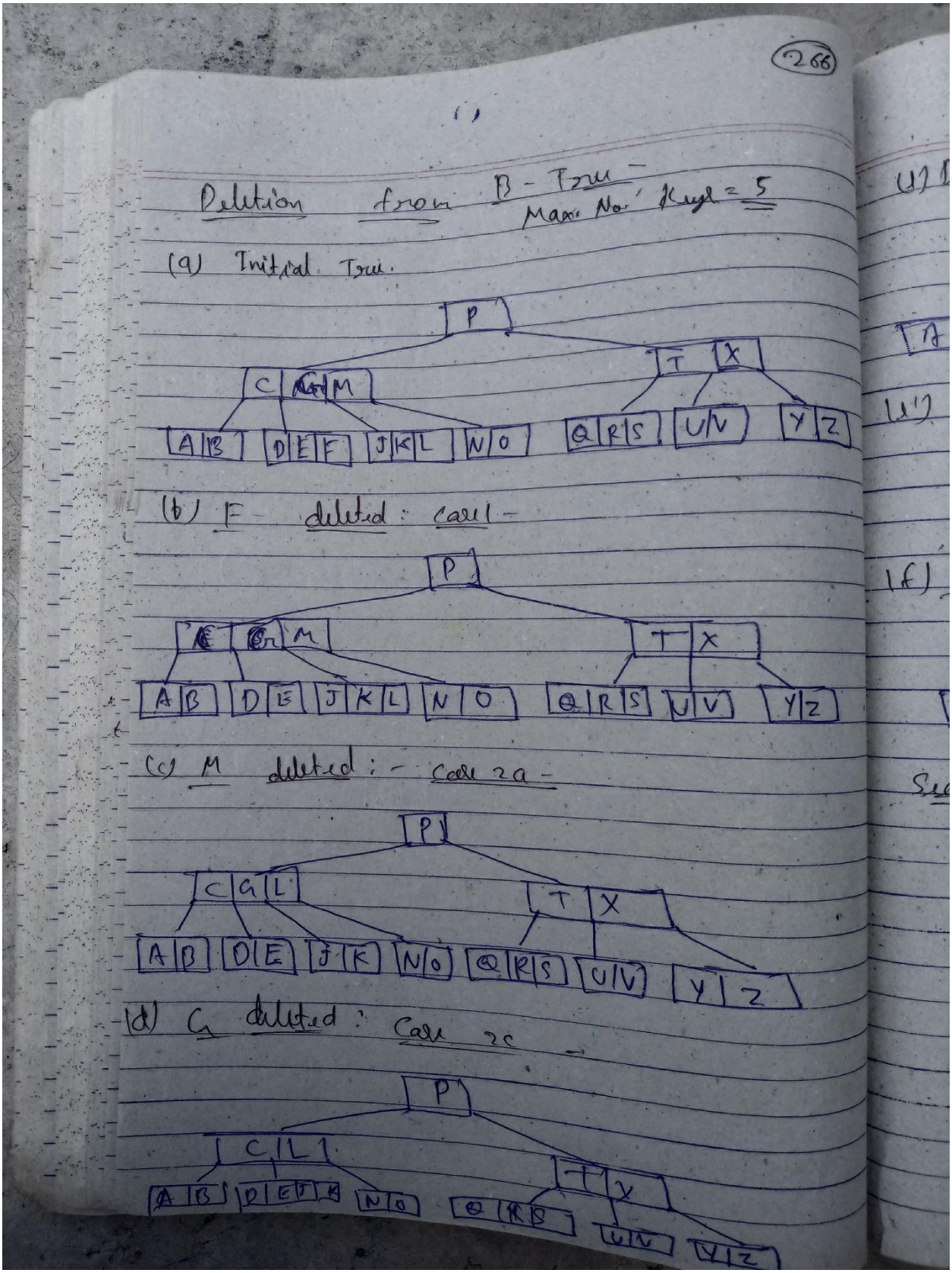

261
Sparse Indexing
Min Disk Read.
id
Search 1B
2 Disk reads
Sorted block
Large Table
Multi level Indexing
id
1 block
Sparse Index
Large Table

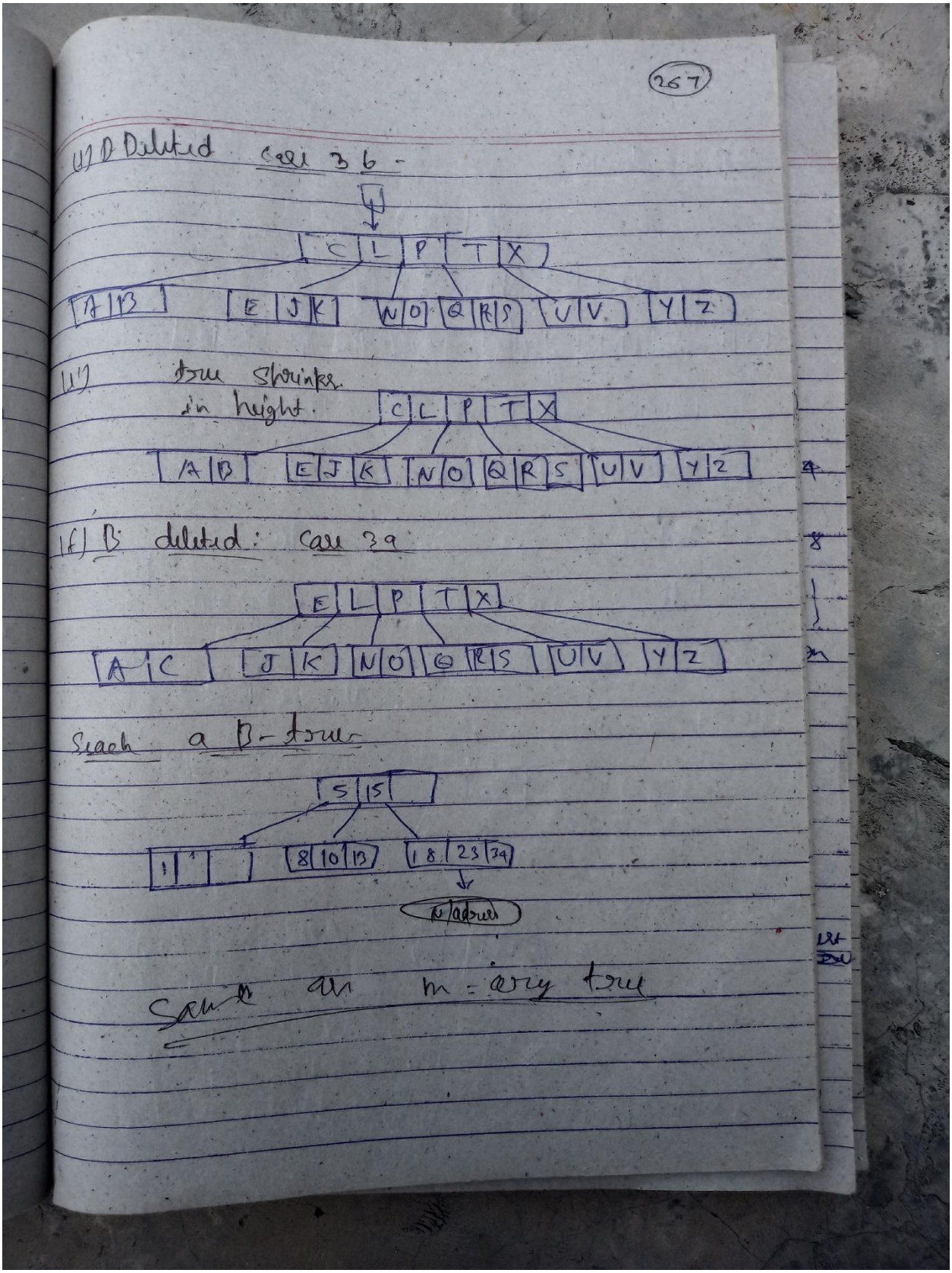

262
Indexed as m-way tree
m-way search tree
m+1 ptr
3.2
Multi-level index as an m-ary search tree
2nd level sparse index
NULL
1st level
B Tree / B+ Tree -
→ inspired from m-ary tree
→ Each node is black
→ rooted
→ All leaf node are at some depth
→ All internal node (except root) must have ⌈m/2⌉ children
→ root has atleast 2 children.

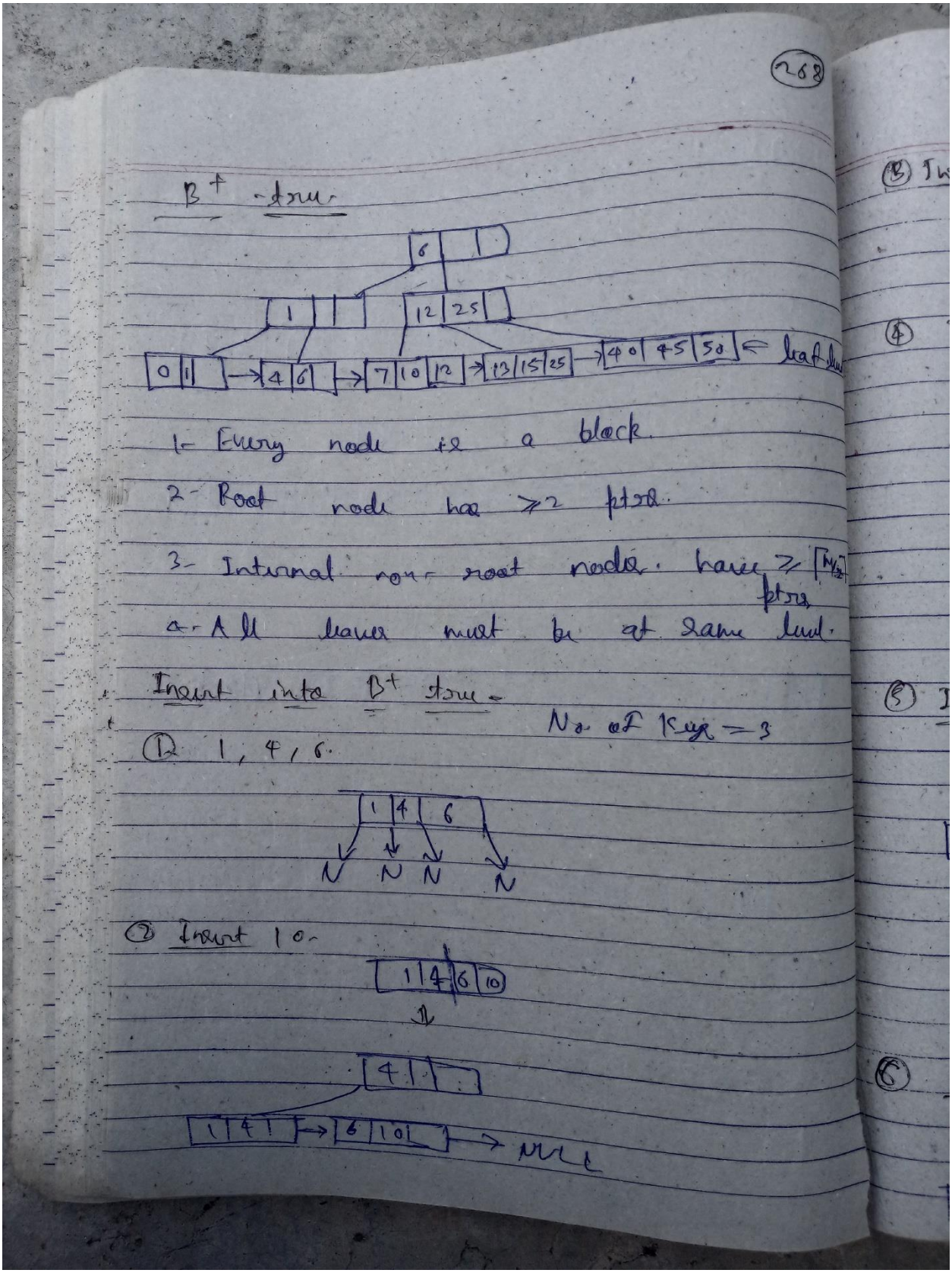

263
node
root
node
Build a B-Tree (insert)
① 5, 10, 15-
root
5 10 15
N N N N
order = 4 = m+1 = max number of pointer in a node
Number of key = 3
② Add 1 :-
1 5 10 15
root
(a)
5
10 15
right biasing
(b)
root
10
1 5
15
left biasing
③ Add 23 to
5
1
10 15 23
fill bottom up

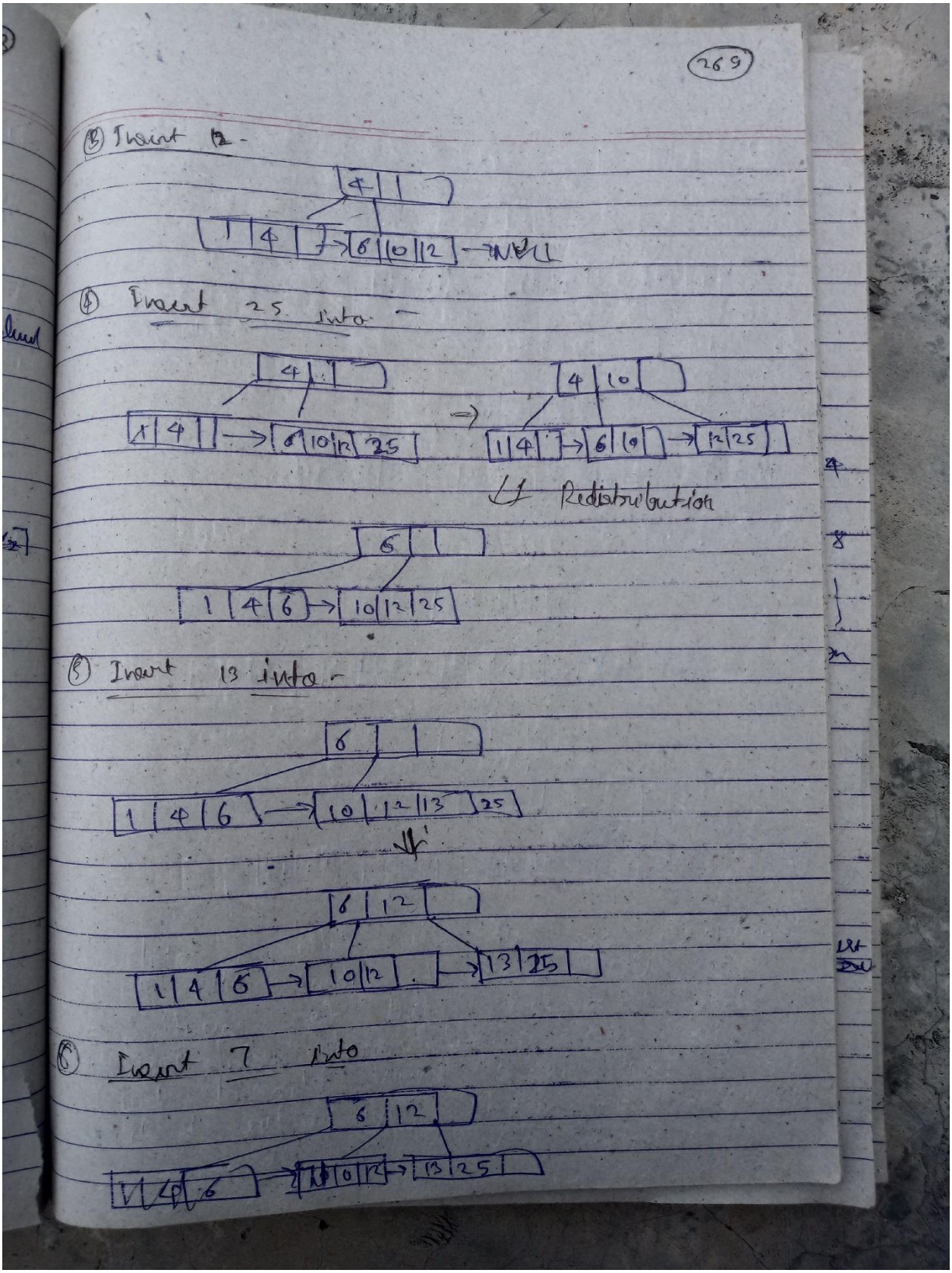

264
④ Add 35 to
root
Redistributing keys to reduce no. of nodes
⑤ add 13 to
Redistribution

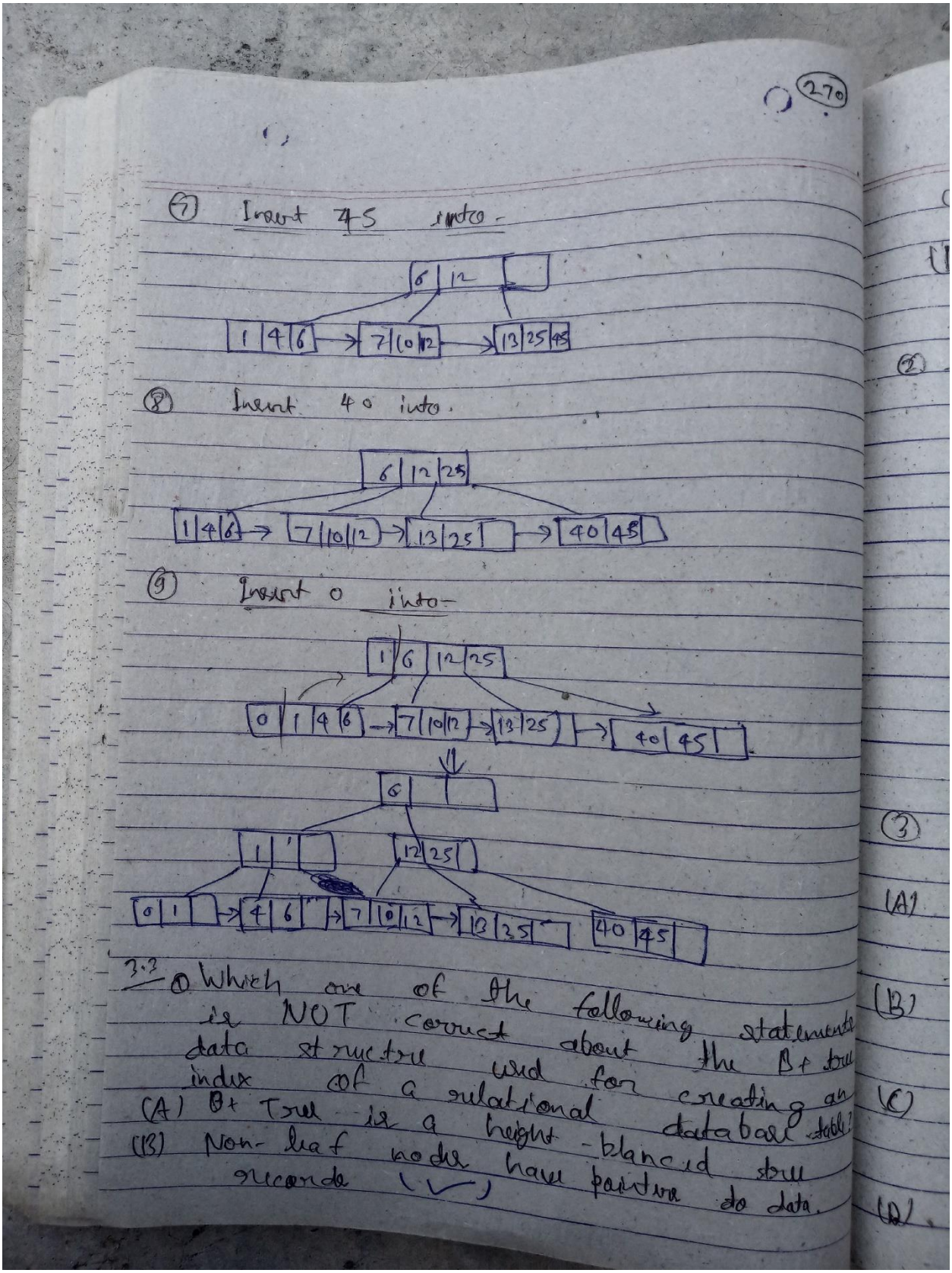

65
6. Add 8 to
13
1 5 8 10 15 23 35
5 13
1
8 10
15 23 35
Redistribution
Not Possible
7. Add 18 to
5 13
1
8 10
15 18 23 35
5 13 18
1
8 10
15 18
23 35
Redistribution
5 15
1
8 10 13
18 23 35


266
Deletion from B - Tree -
Max. No. Keys = 5
(a) Initial Tree.
P
C G M
T X
A B D E F J K L N O Q R S U V Y Z
(b) F deleted : Case I -
P
C G M
T X
A B D E J K L N O Q R S U V Y Z
(c) M deleted :- Case 2a -
P
C G L
T X
A B D E J K N O Q R S U V Y Z
(d) G deleted : Case 2c -
P
C L
T X
A B D E J K N O Q R S U V Y Z


267
(1) D Deleted case 3b -
C L P T X
A B
E J K
N O
Q R S
U V
Y Z
(2) tree shrinks in height.
C L P T X
A B
E J K
N O
Q R S
U V
Y Z
(3) B deleted: case 3a
E L P T X
A C
J K
N O
Q R S
U V
Y Z
Search a B-tree
5 15
1
8 10 13
18 23 39
Address
Search an m-way tree


B+ tree
6
1
12 25
0 1
4 6
7 10 12
13 15 25
40 45 50
leaf
1- Every node is a block.
2- Root node has ≥ 2 ptrs.
3- Internal non-root nodes have ≥ ⌈M/2⌉ ptrs
a- All leaves must be at same level.
Insert into B+ tree -
No. of keys = 3
(1) 1, 4, 6.
1 4 6
N N N N
(2) Insert 10.
1 4 6 10
4
1 4
6 10
NULL


269
③ Insert 12 -
4
1 4 6 10 12 NULL
④ Insert 25 into -
4
1 4 6 10 12 25
4 10
1 4 6 10 12 25
Redistribution
6
1 4 6 10 12 25
⑤ Insert 13 into -
6
1 4 6 10 12 13 25
6 12
1 4 6 10 12 13 25
⑥ Insert 7 into
6 12
1 4 6 7 10 12 13 25

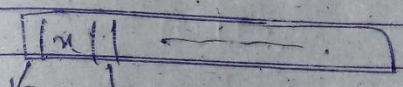

270
(7) Insert 45 into.
6 12
1 4 6
7 10 12
13 25 45
(8) Insert 40 into.
6 12 25
1 4 6
7 10 12
13 25
40 45
(9) Insert 0 into.
1 6 12 25
0 1 4 6
7 10 12
13 25
40 45
6
1
12 25
0 1
4 6
7 10 12
13 25
40 45
3.3 Which one of the following statements is NOT correct about the B+ tree data structure used for creating an index of a relational database table?
(A) B+ Tree is a height-balanced tree
(B) Non-leaf nodes have pointers to data records (✓)

271

(C) Key values in each node are kept in sorted order

(D) Each leaf node has a pointer to the next leaf node.

(2) In a B+ tree, if the search-key value is 8 bytes long, the block size is 512 bytes and the block pointer is 2 bytes, then the maximum order of the B+ tree is $(m+1)=52$



m keys
order: m+1 pointers

$8m + 2(m+1) \leq 512$

$10m + 2 \leq 518$

$10m \leq 510$

$m \leq 51$

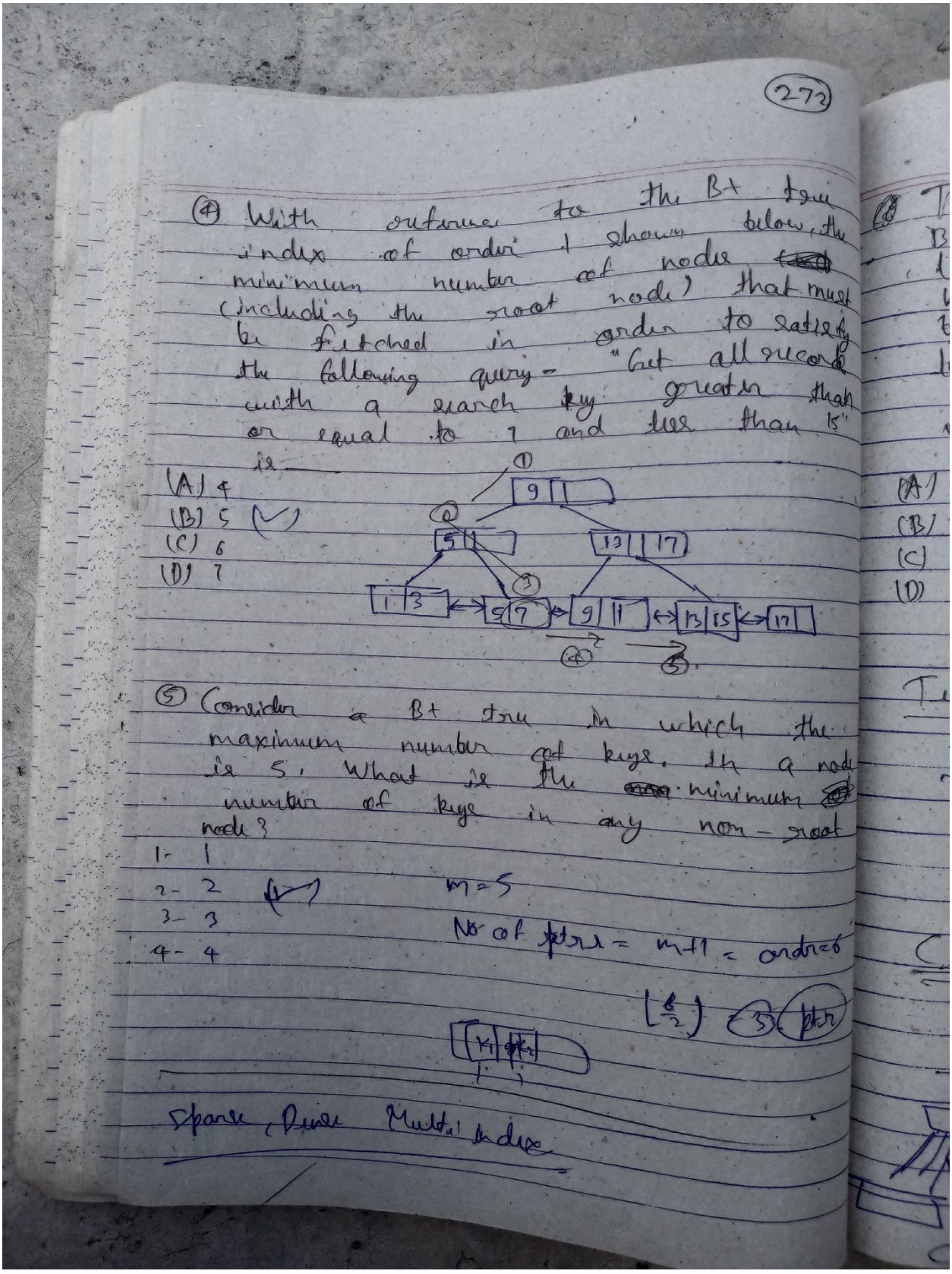
(3) B+ Trees are considered BALANCED because -

(A) the length of the paths from the root to all leaf nodes are all equal. (✓)

(B) the lengths of the paths from the root to all leaf nodes differ from each other by at most 1.

(C) the number of children of any two non-leaf sibling nodes differ by at most 1.

(D) the number of records in any two leaf nodes differ by at most 1.



272

④ With reference to the B+ tree index of order 1 shown below, the minimum number of nodes (including the root node) that must be fetched in order to satisfy the following query - "Get all records with a search key greater than or equal to 7 and less than 15" is

(A) 4
(B) 5 (✓)
(C) 6
(D) 7

⑤ Consider a B+ tree in which the maximum number of keys is 5. What is the minimum number of keys in any non-root node?

1- 1
2- 2 (✓)
3- 3
4- 4

m = 5

No of ptrs = m+1 = order

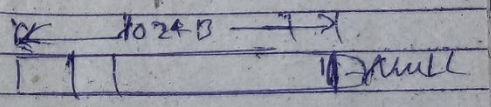
$\left(\frac{6}{2}\right)$ 3 ptr

Sparse, Dense, Multi index

(273)

Q The order of a leaf node in a B+ tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node?

(A) 63
(B) 64
(C) 67
(D) 68



$(9+7)m + 6 \leq 1024$

$16m \leq 1018$

$m \leq 63$

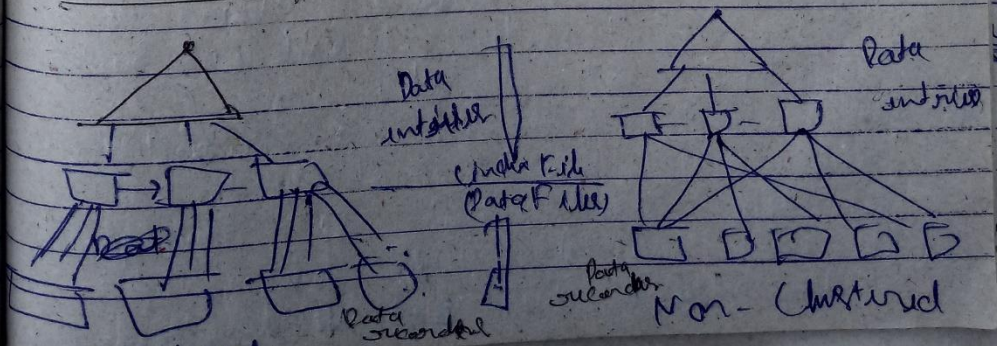
Terminology related to indexes -

Primary & Secondary Index

key, non-key.

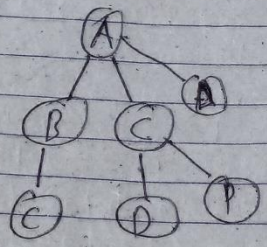
key Primary key

Clustered vs Unclustered Index -



27

Ternary Search Tree

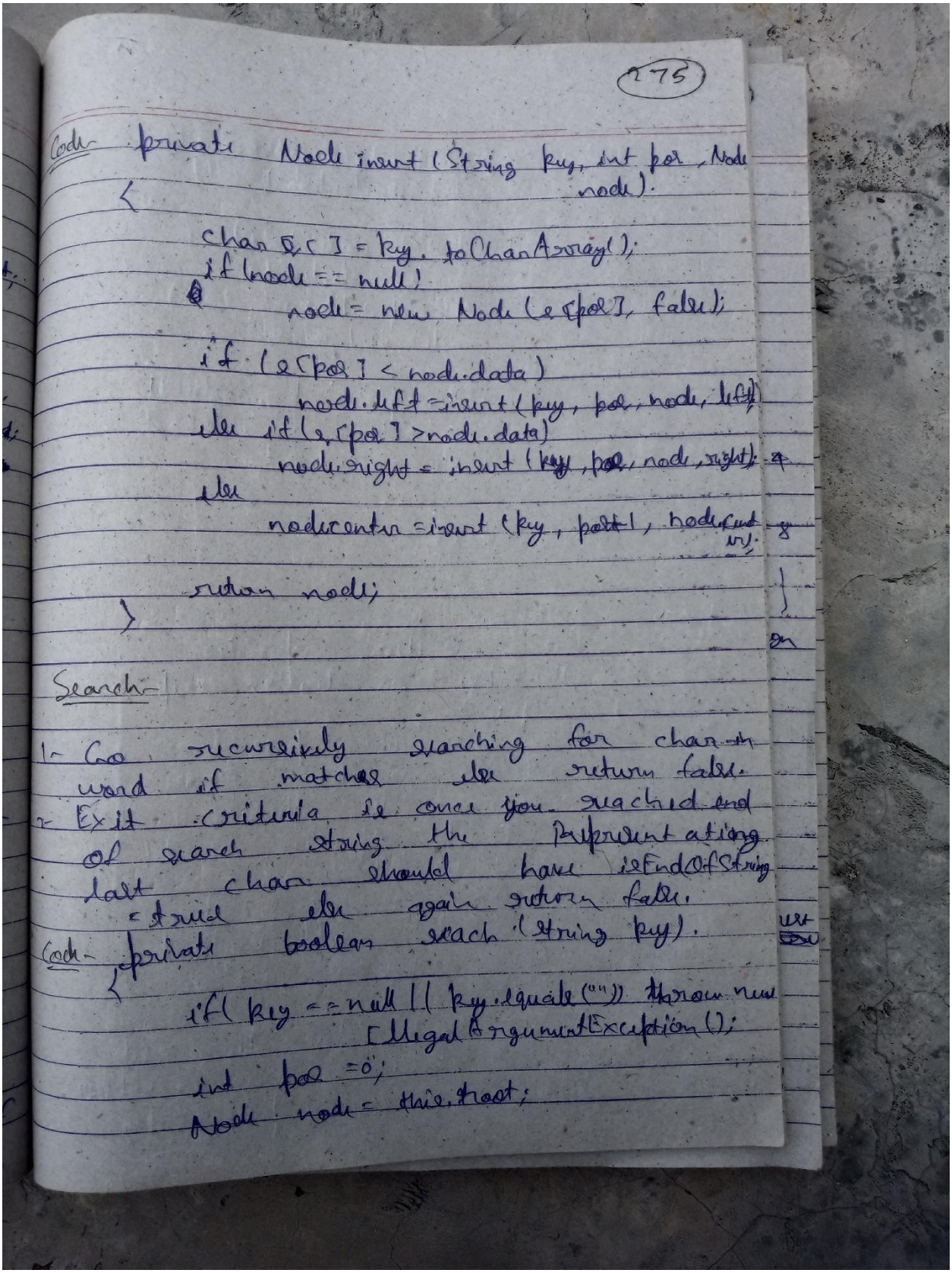


Code Class Node

```
{
    char data;
    boolean isEnd;
    Node left, center, right;
    public Node (char data,
                 boolean isEnd)
    {
        this.data = data;
        this.isEnd = isEnd;
        left = right = center
             = null;
    }
}
```

Insertion.

1- If Tree is empty, then insert the element.
2- If current character of word is smaller than root's character, then insert this word in left subtree of root.
3- If current character of word is greater than root's character, then insert this word in right subtree of root.
4- If current character of word is same as root's character,
   (a) Insert character as root.equal.
   (b) Check if the last character of the word, then set isEndOfString = true



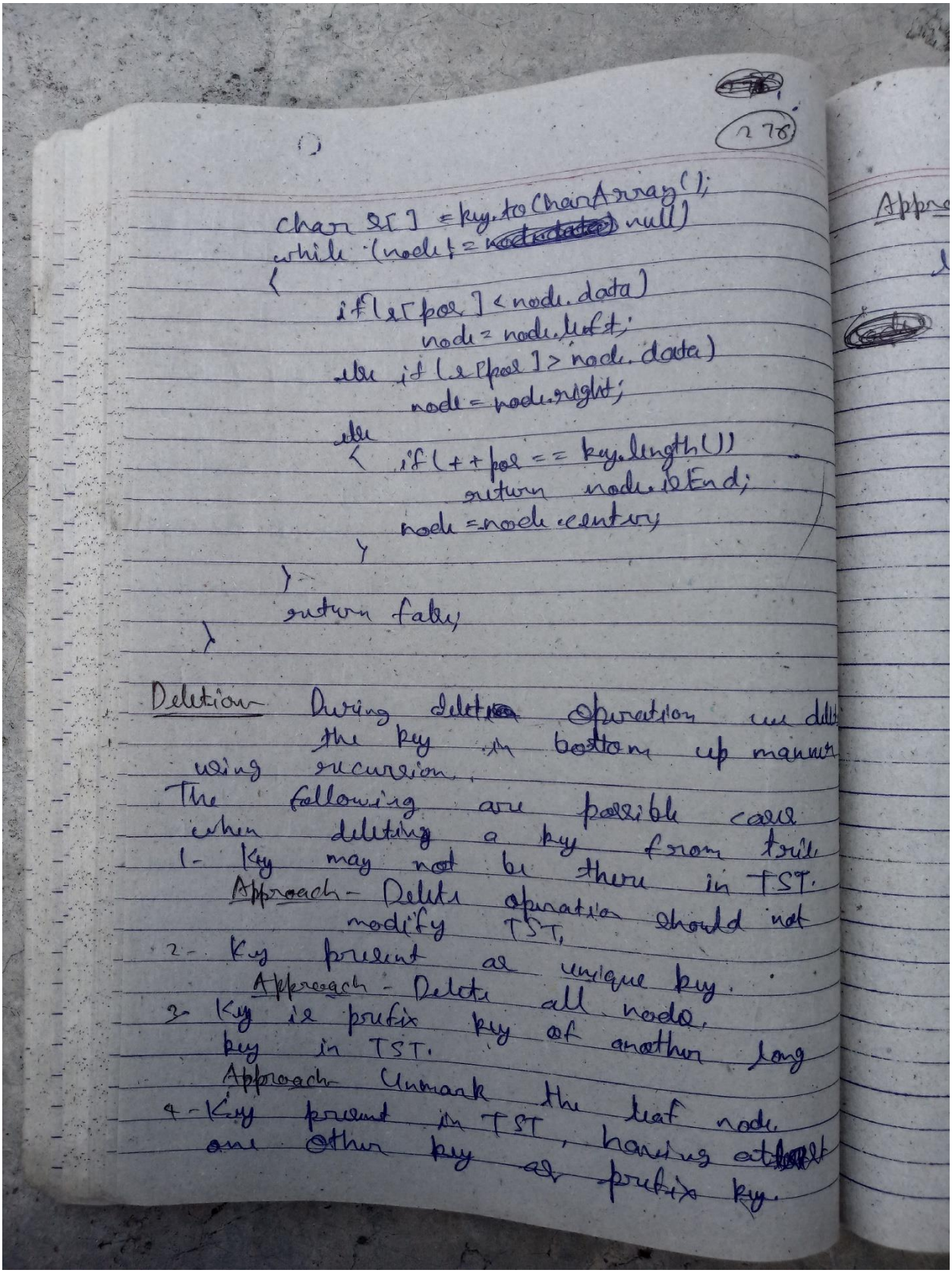
175

Code- private Node insert (String key, int pos, Node node).
{
char a[] = key.toCharArray();
if (node == null)
node = new Node (a[pos], false);
if (a[pos] < node.data)
node.left = insert (key, pos, node.left);
else if (a[pos] > node.data)
node.right = insert (key, pos, node.right);
else
node.centre = insert (key, pos+1, node.centre);
return node;
}

Search-

1- Go recursively searching for char of word if matches else return false.
2- Exit criteria is once you reached end of search string the current string last char should have isEndOfString = true else again return false.

Code- private boolean search (String key).
{
if (key == null || key.equals("")) throw new IllegalArgumentException();
int pos = 0;
Node node = this.root;



```
char s[] = key.toCharArray();
while (node != null)
{
    if (s[pos] < node.data)
        node = node.left;
    else if (s[pos] > node.data)
        node = node.right;
    else
    {
        if (++pos == key.length())
            return node.isEnd;
        node = node.center;
    }
}
return false;
}
```

Deletion During deletion operation we delete the key in bottom up manner using recursion.

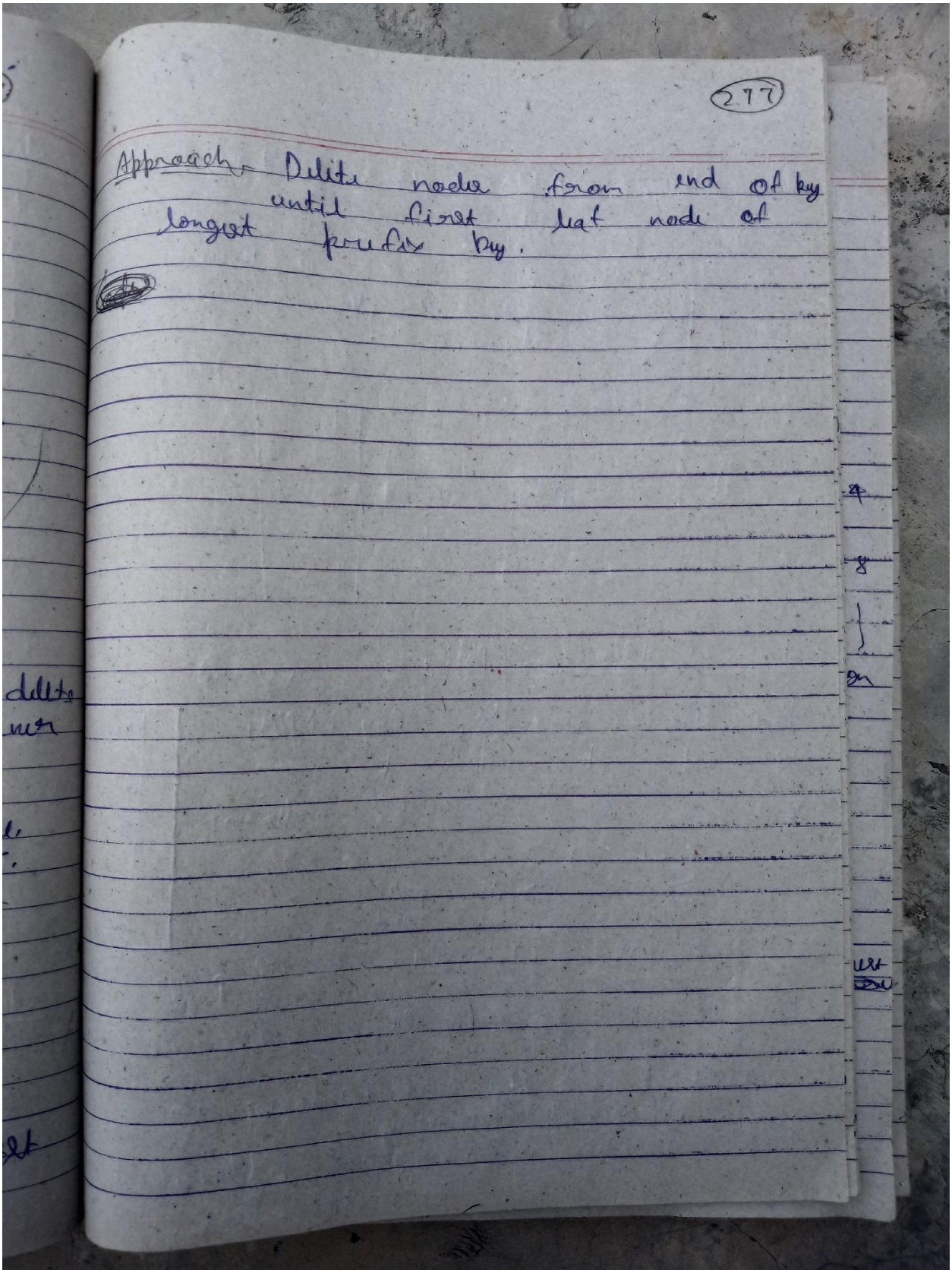
The following are possible cases when deleting a key from trie.

1- Key may not be there in TST.
Approach - Delete operation should not modify TST.

2- Key present as unique key.
Approach - Delete all nodes.

3- Key is prefix key of another long key in TST.
Approach - Unmark the leaf node.

4- Key present in TST, having atleast one other key as prefix key.


2.77
Approach - Delete nodes from end of key until first last node of longest prefix key.

(278)

5.1 Page Replacement Strategies –

Page reference string – { L: length, n: # unique Pages (Sequence of reference) }

Pr: VA₁, VA₂, VA₃ — VA₁ — → time.

Sequence page numbers

4, 5, 5, 4

Page fault — Pure demand page.

P1 — f0
f1
P5 — f2
f3
f4

Main memory.

Frame Allocation Policy

(VMM)

$P_{r1}, P_{r2}, P_{r3}, P_{r4}, P_{r5}$: MP/MT

$S_1 \quad S_2 \quad S_3 \quad S_4 \quad S_5$: demand

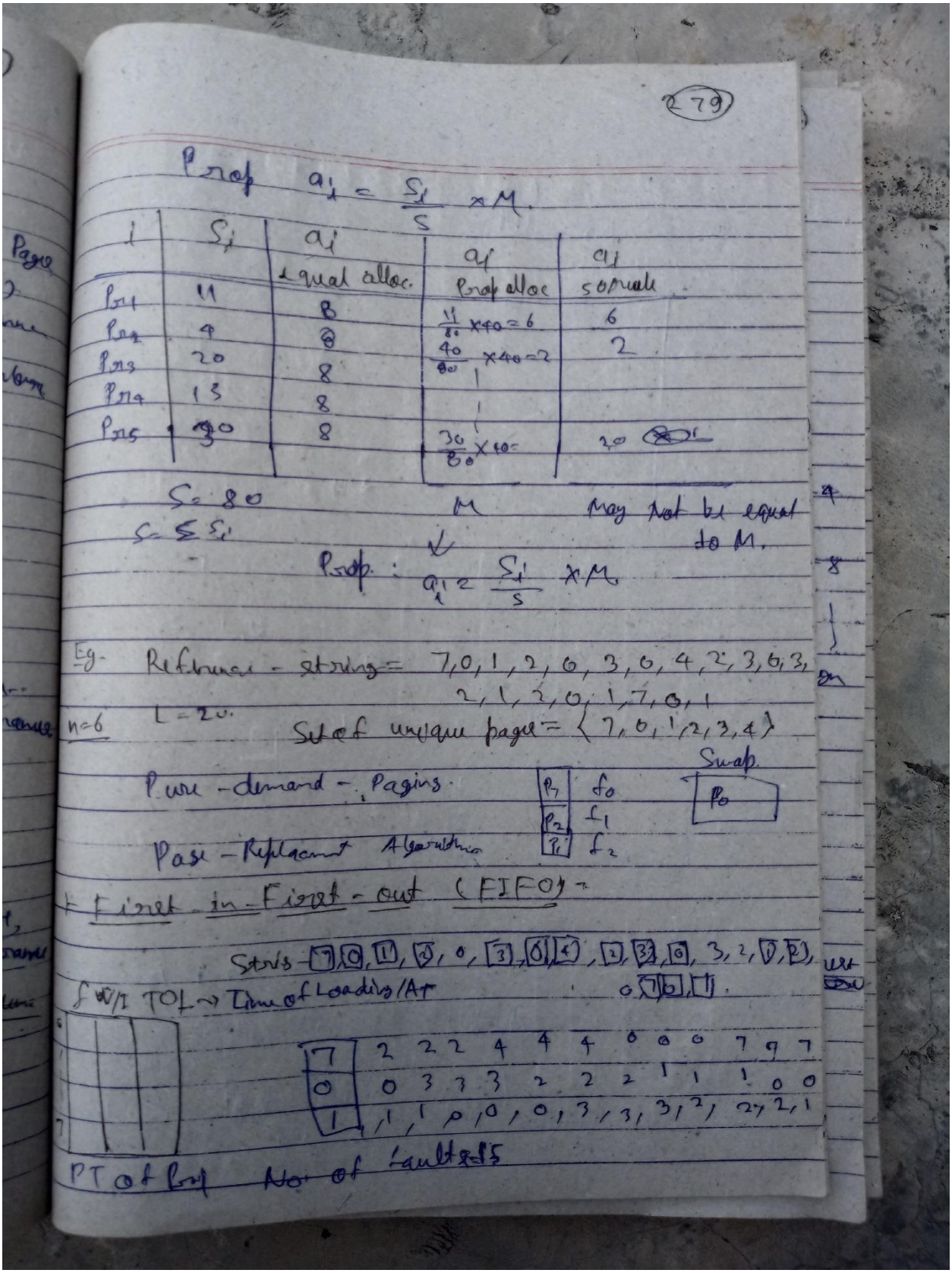
$\sum S_i = S > M$ — RAP used

0 frame, 1, 2, fi, …, M-1 frame — Physical Memory

Let $P_{r1}, P_{r2}, P_{r3}, P_{r4}, P_{r5}$

M = 40, = No. of frames in Physical Memory

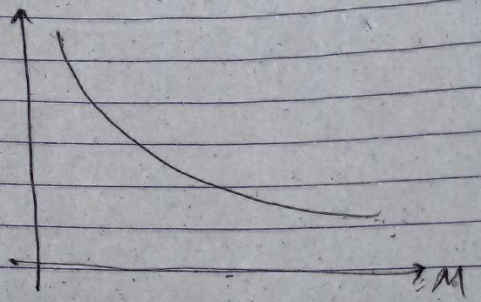
a_i = No. of frames allocated to P_{ri}


79
Prof $a_i = \frac{S_i}{S} \times M$
i | S_i | a_i equal alloc. | a_i Prop alloc | a_i corrected
P1 | 11 | 8 | $\frac{11}{80} \times 40 = 6$ | 6
P2 | 4 | 8 | $\frac{4}{80} \times 40 = 2$ | 2
P3 | 20 | 8 | |
P4 | 13 | 8 | |
P5 | 30 | 8 | $\frac{30}{80} \times 40 =$ | 20
S = 80
S = Σ Si
M
May not be equal to M.
Prop: $a_i = \frac{S_i}{S} \times M$
Eg. Reference string = 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
n=6 L=20.
Set of unique page = {7,0,1,2,3,4}
Pure-demand-Paging
Page-Replacement Algorithm
Swap.
P0
P1 f0
P2 f1
P3 f2
First-in-First-out (FIFO)
Strs 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
F/I TOL → Time of Loading/AT
7 2 2 2 4 4 4 0 0 0 7 7 7
0 0 3 3 3 2 2 2 1 1 1 0 0
1 1 1 0 0 0 3 3 3 2 2 2 1
PT of Prog No. of faults

280

M = No. of frame = 4
No. of page fault = 10

Page fault rate.



Eg ref. string = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

(a) M = 3
No. of Page fault = 9

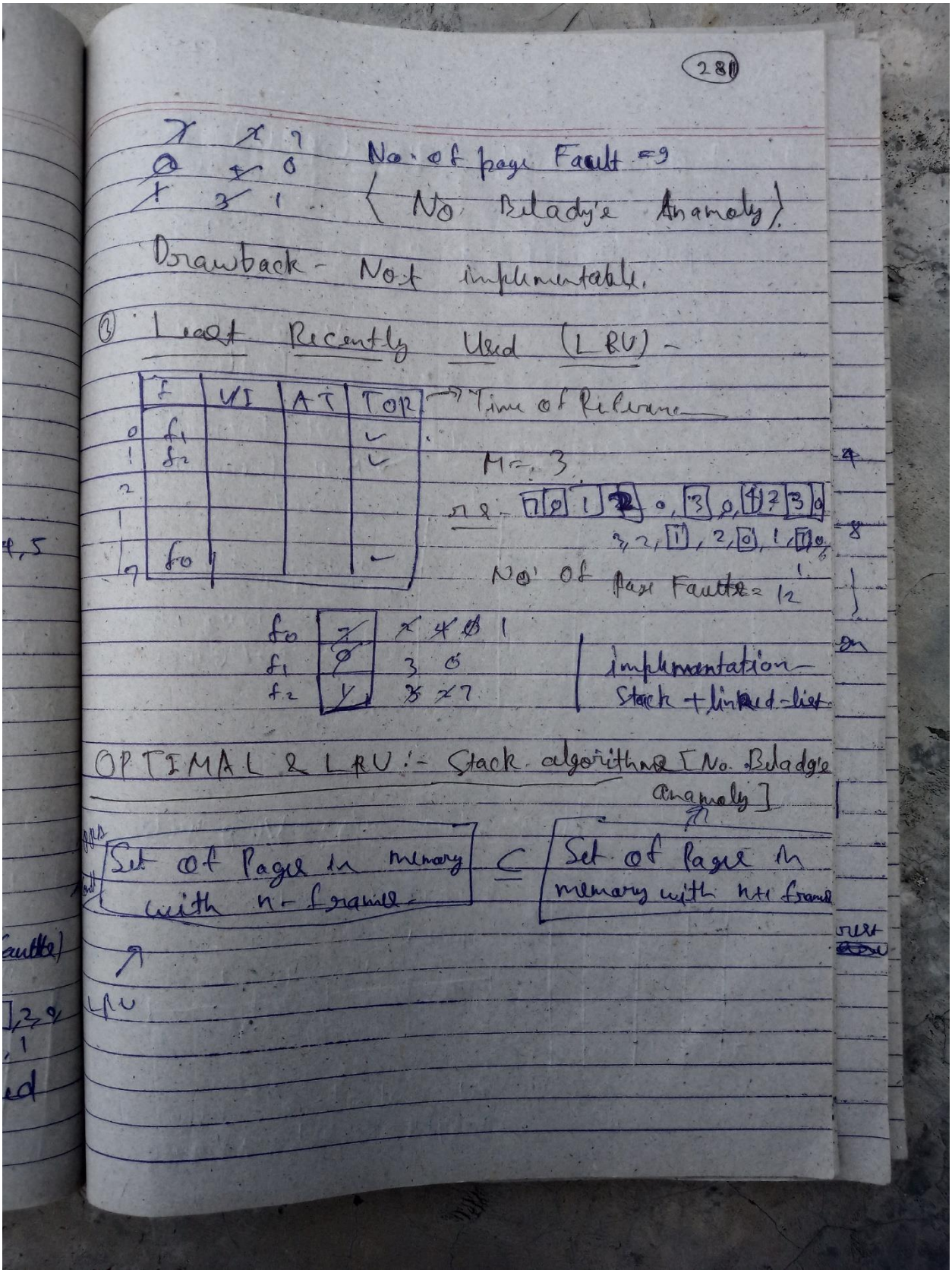
(b) M = 4, No. of Page fault = 10

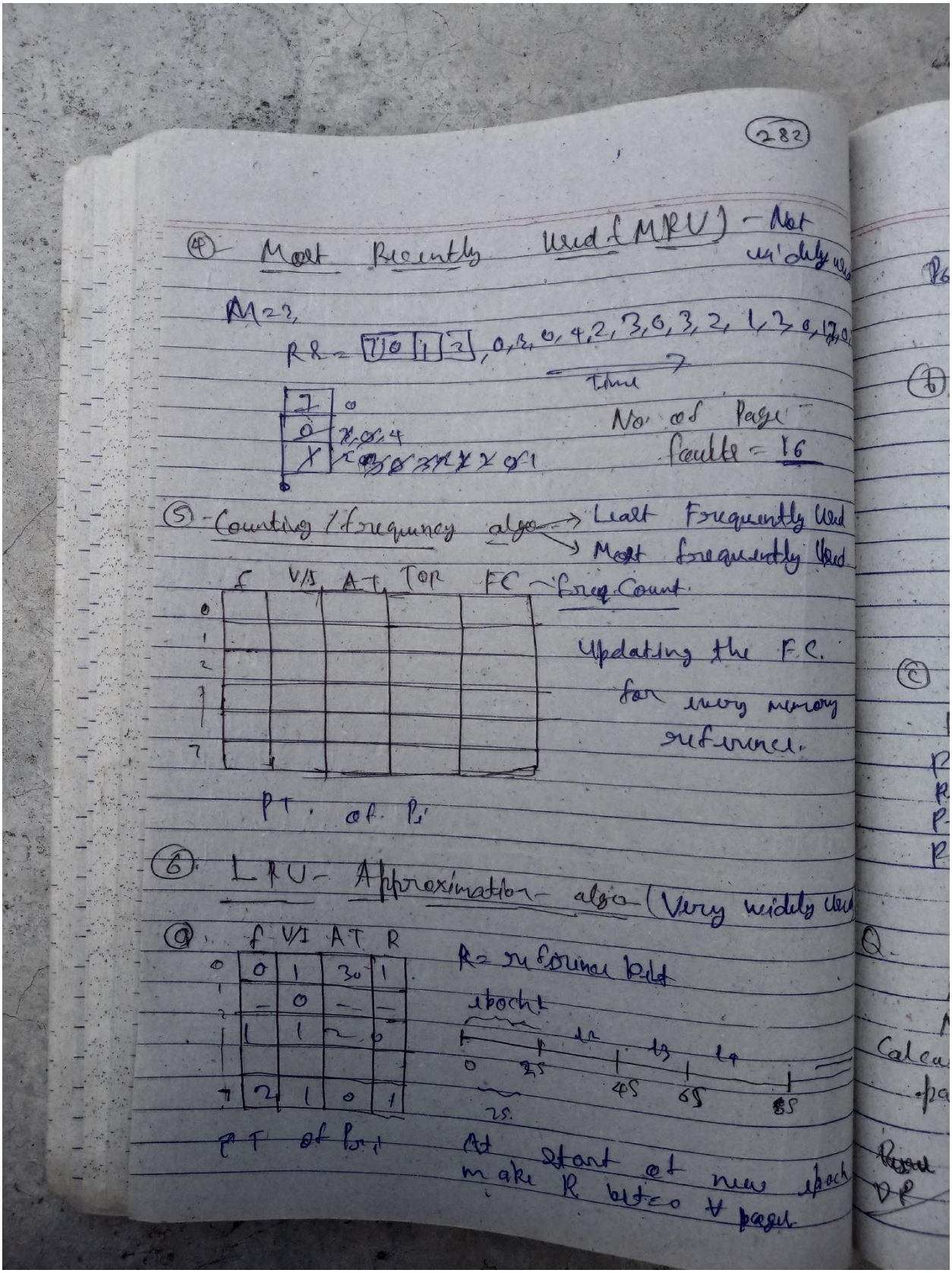
Belady's Anamoly

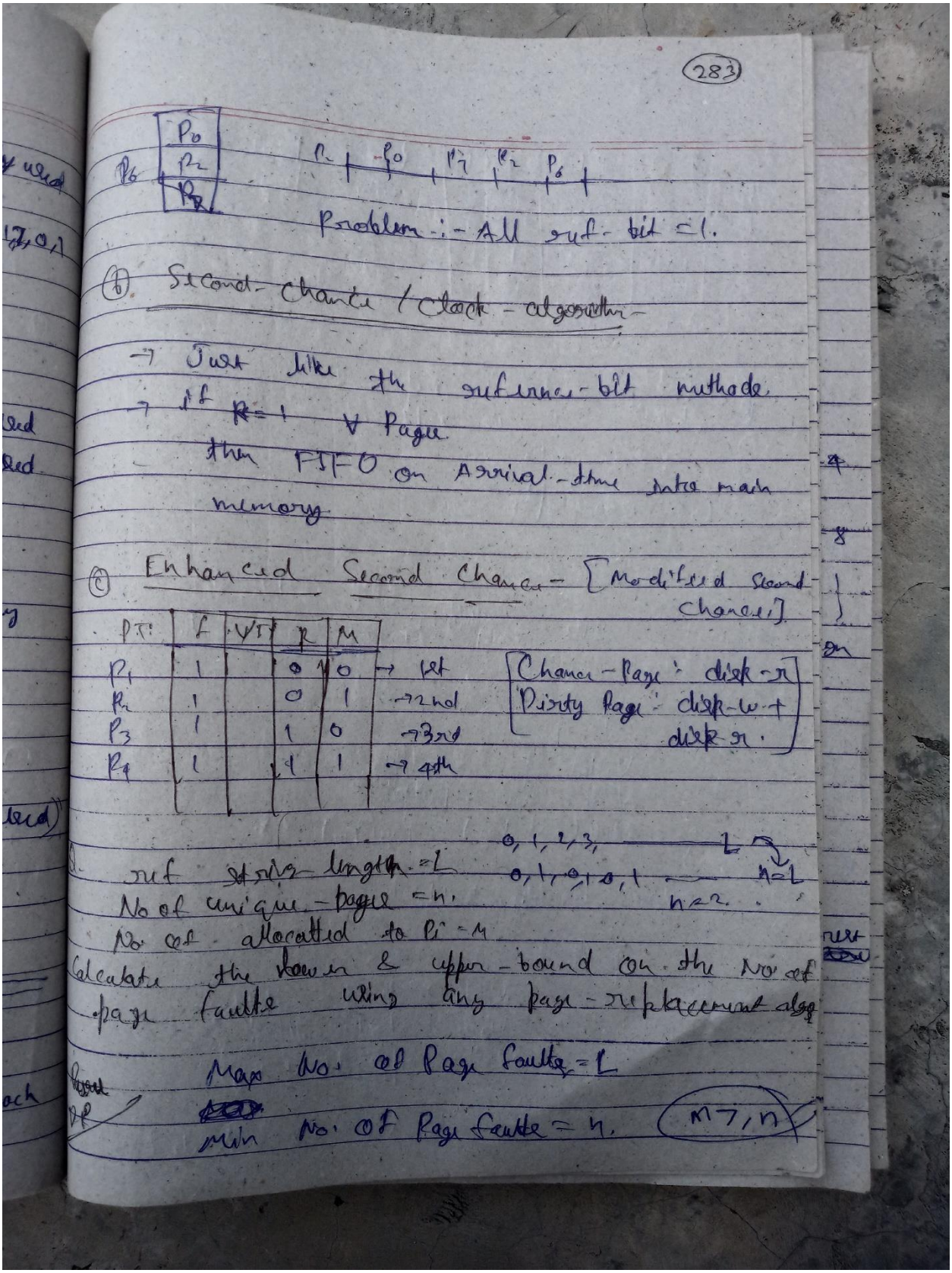
OPTIMAL replacement - (Min No of Page faults)
Benchmark

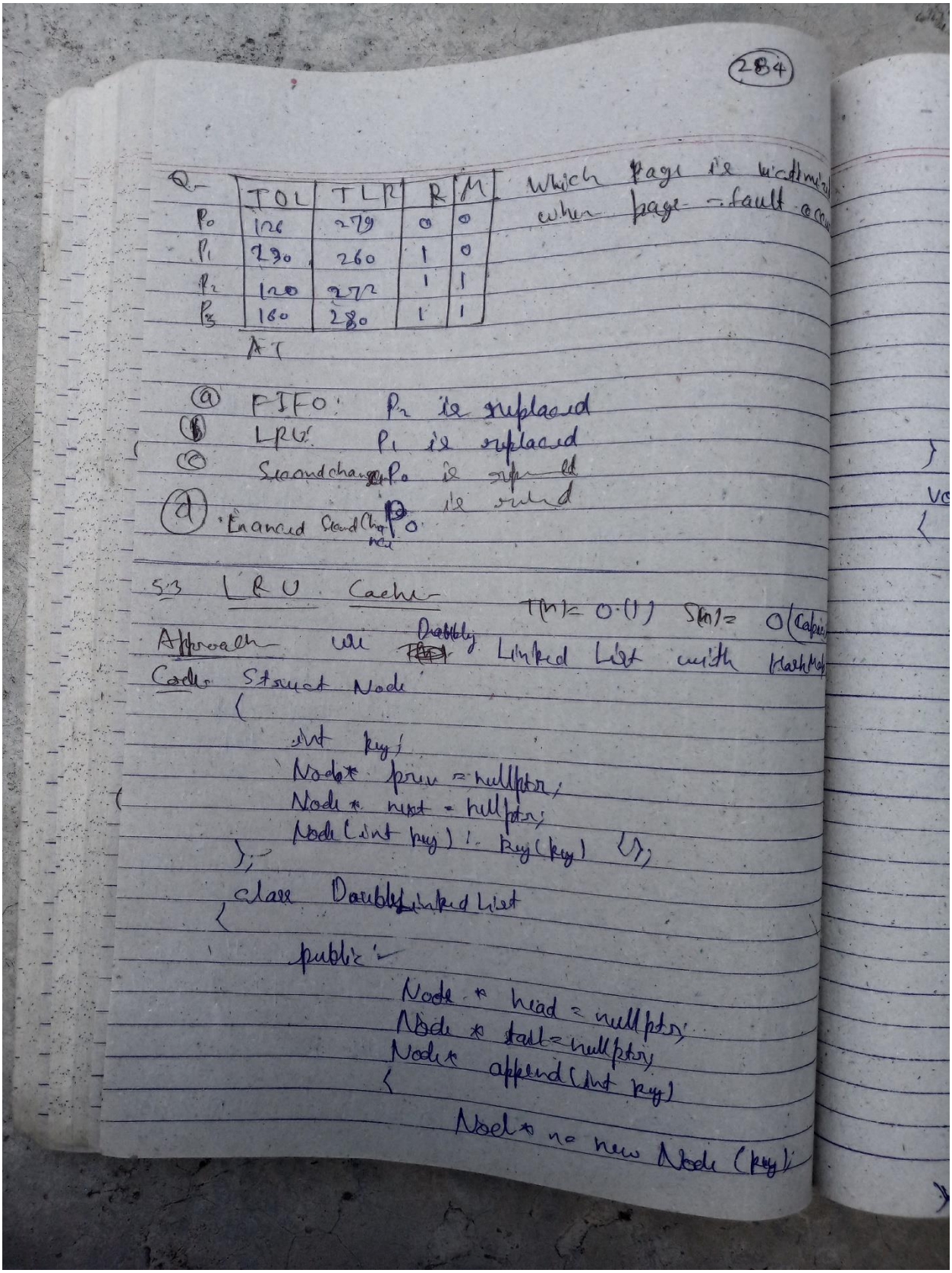
M = 3 R.S = 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Replace the page that not needed longest into the future


No. of page Fault = 9
{ No Belady's Anamoly }
Drawback - Not implementable.
③ Least Recently Used (LRU) -
F
VI
AT
TOR
Time of Reference
f0
f1
f2
M = 3
No. of Page Faults = 12
implementation - Stack + linked-list
OPTIMAL & LRU :- Stack algorithms [No. Belady's Anamoly]
Set of Page in memory with n-frames
Set of Page in memory with n+1 frame
LRU


282
④ Most Recently Used (MRU) - Not widely used
M=3,
RS = 7,0,1,2,0,4,2,3,0,3,2,1,2,0,1,7,0,1
Time →
No. of Page faults = 16
⑤ Counting / Frequency algo → Least Frequently Used
→ Most frequently Used
f V/S AT TOP FC - Freq. Count
Updating the F.C. for every memory reference.
PT. of P1
⑥ LRU - Approximation algo (Very widely used
f V/I AT R
R = reference bit
epoch
0 25 45 65 85
25.
PT of P1
At start of new epoch make R bit = 0 of page


283
Problem :- All ref. bit = 1.
(b) Second Chance / Clock algorithm
→ Just like the reference bit methods
→ if R=1 ∀ Page
then FIFO on Arrival time into main memory
(c) Enhanced Second Chance - [Modified Second Chance]
Chance-Page : disk-r
Dirty Page : disk-w + disk-r.
No of unique page = n.
No of allocated to Pi = M
Calculate the lower & upper bound on the No of page faults using any page replacement algo
Max No. of Page faults = L
Min No. of Page faults = n.



284

Q-

	TOL	TLR	R	M
P_0	126	279	0	0
P_1	230	260	1	0
P_2	120	272	1	1
P_3	160	280	1	1

Which page is victimized when page fault occurs

A:

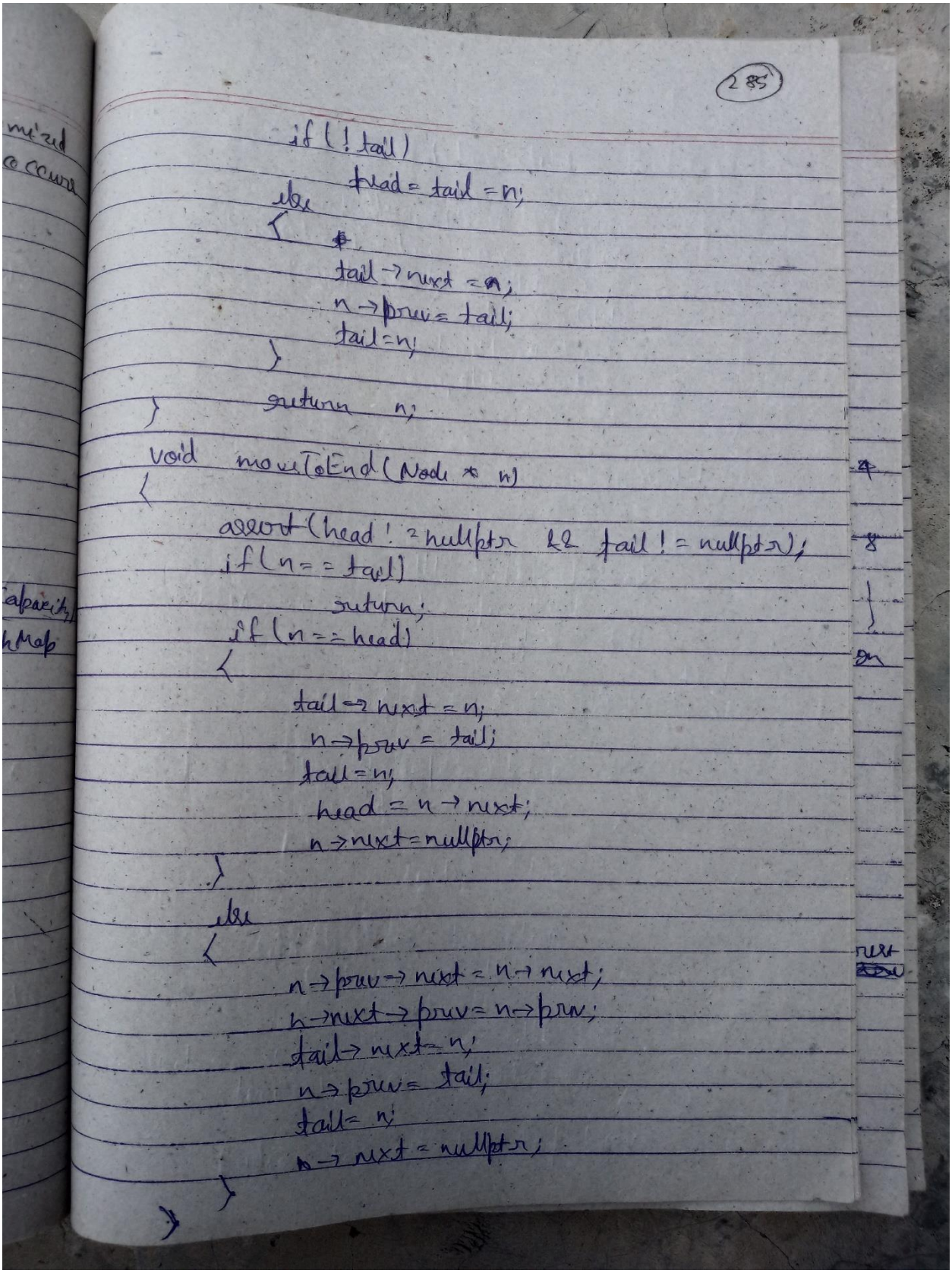
(a) FIFO: P_2 is replaced
(b) LRU: P_1 is replaced
(c) Second chance: P_0 is replaced
(d) Enhanced Second Chance: P_0

5.3 LRU Cache — T(n) = O(1) S(n) = O(Capacity)

Approach — We will use Doubly Linked List with HashMap

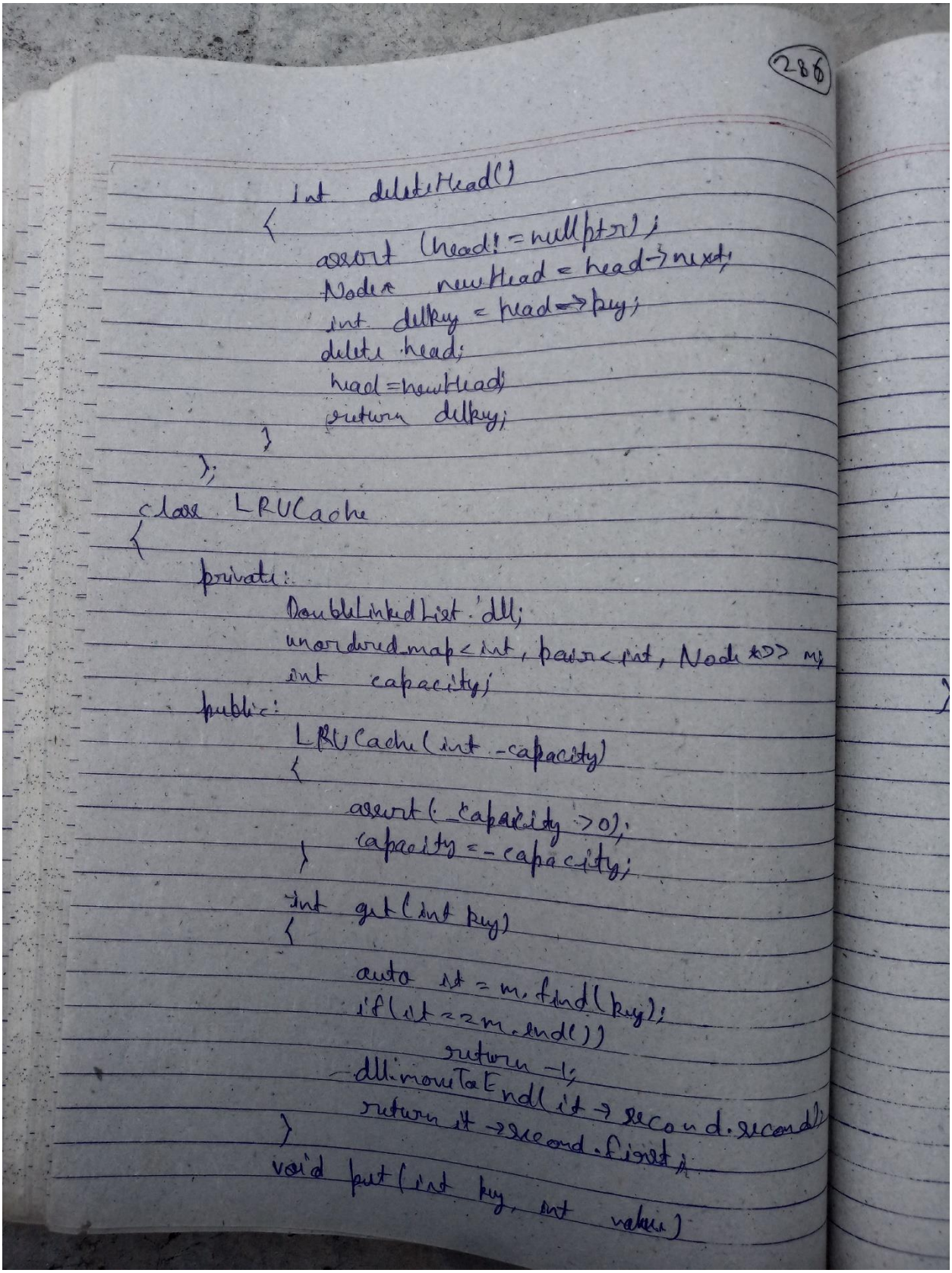
Code:

```
Struct Node
{
    int key;
    Node* prev = nullptr;
    Node* next = nullptr;
    Node(int key) : key(key) {};
};
class DoublyLinkedList
{
public:
    Node* head = nullptr;
    Node* tail = nullptr;
    Node* append(int key)
    {
        Node* n = new Node(key);
```

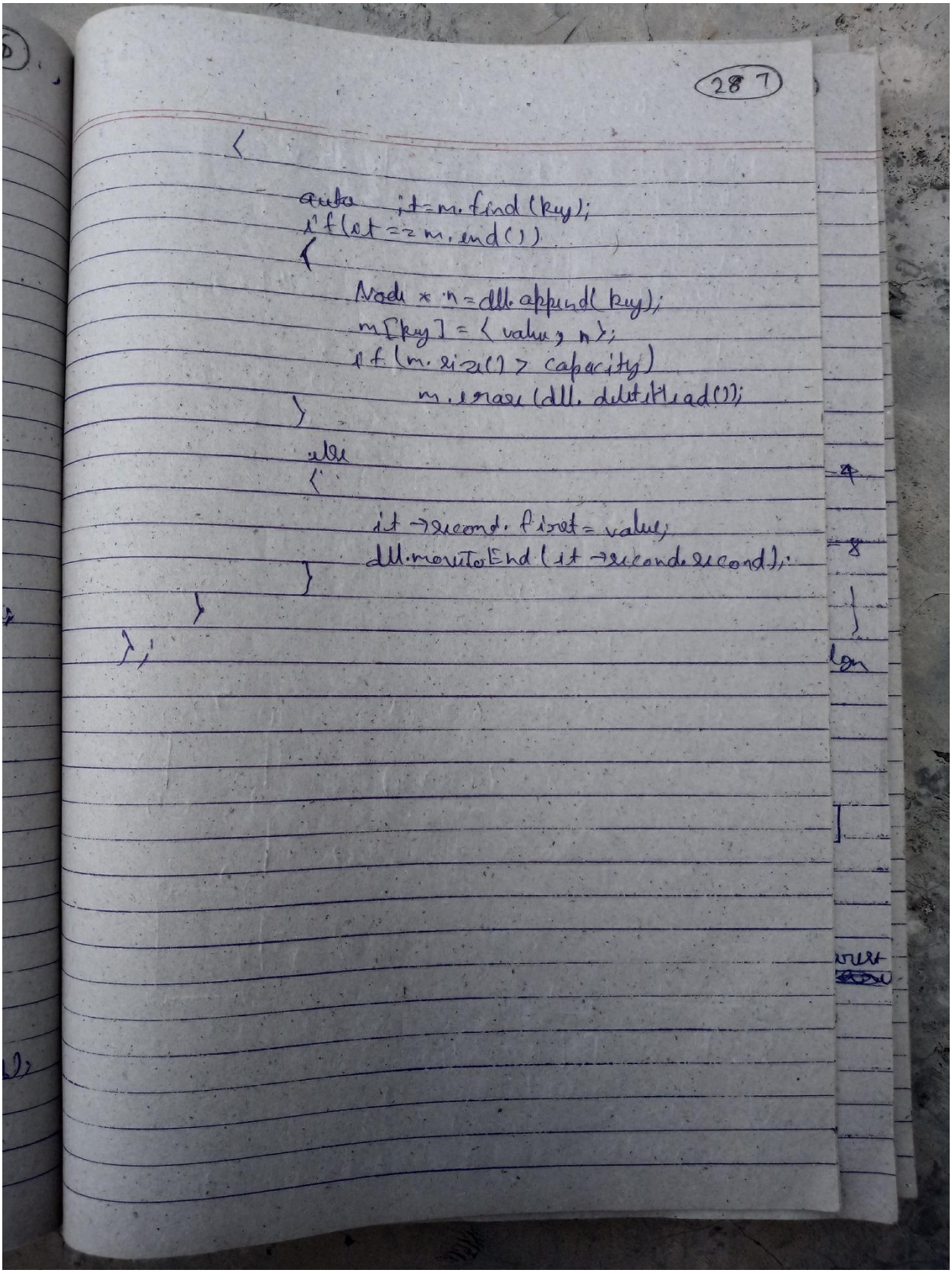


285

```
        if (!tail)
            head = tail = n;
        else
        {
            tail->next = n;
            n->prev = tail;
            tail = n;
        }
        return n;
}
void moveToEnd(Node * n)
{
    assert(head != nullptr && tail != nullptr);
    if(n == tail)
        return;
    if(n == head)
    {
        tail->next = n;
        n->prev = tail;
        tail = n;
        head = n->next;
        n->next = nullptr;
    }
    else
    {
        n->prev->next = n->next;
        n->next->prev = n->prev;
        tail->next = n;
        n->prev = tail;
        tail = n;
        n->next = nullptr;
    }
}
```

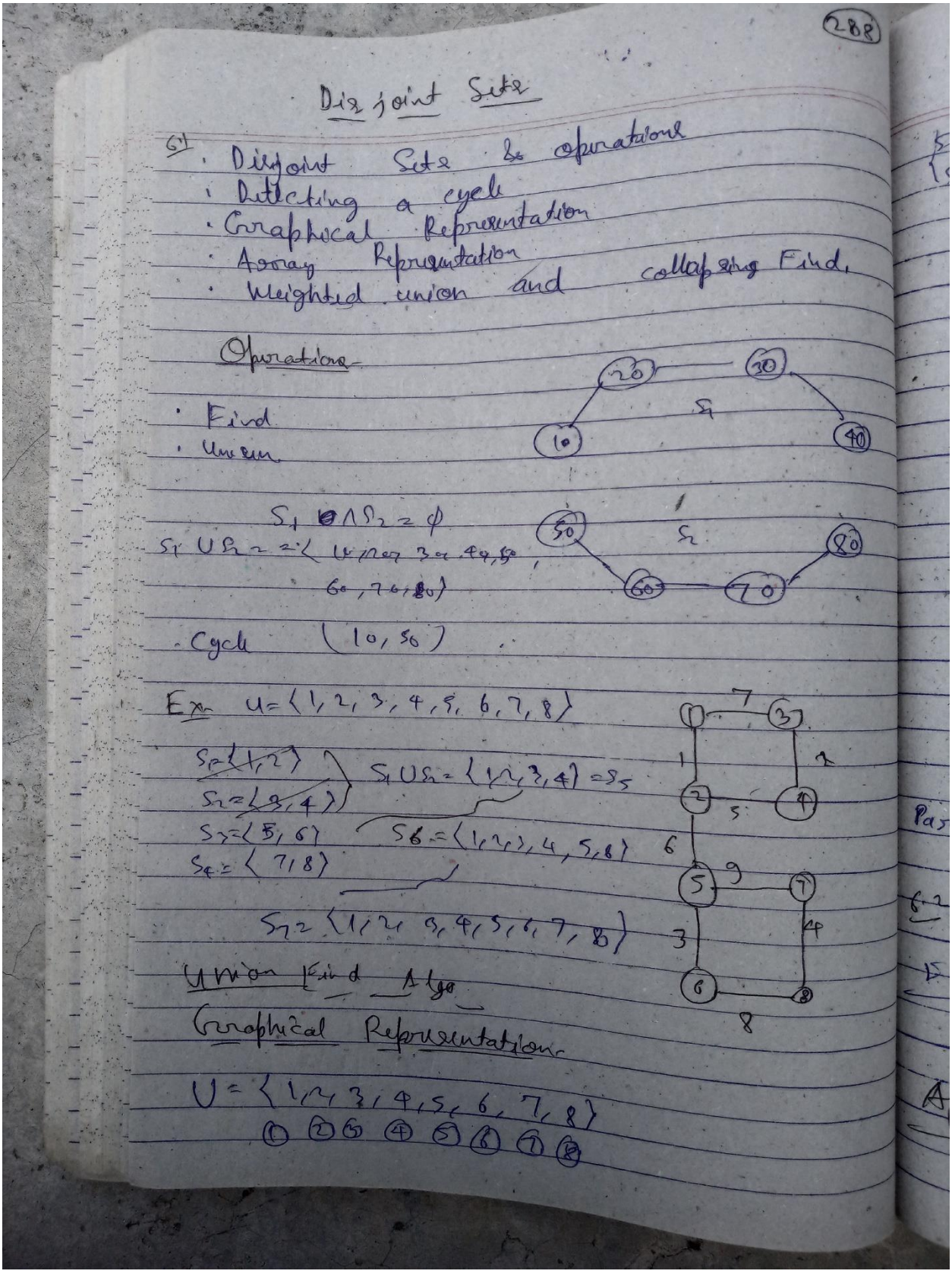


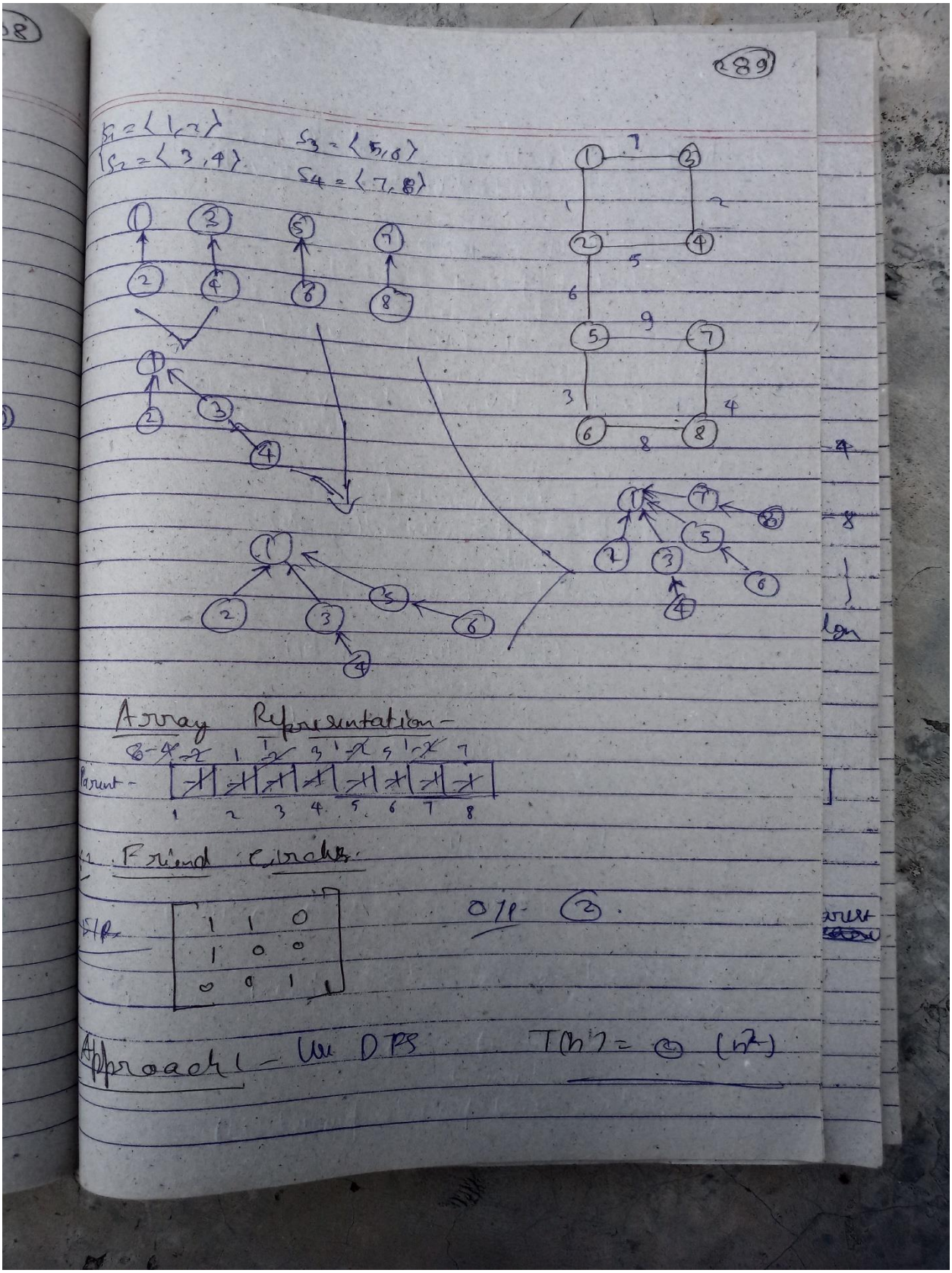
```
int deleteHead()
{
    assert(head != nullptr);
    Node* newHead = head->next;
    int delKey = head->key;
    delete head;
    head = newHead;
    return delKey;
}
};
class LRUCache
{
private:
    DoublyLinkedList dll;
    unordered_map<int, pair<int, Node*>> m;
    int capacity;
public:
    LRUCache(int _capacity)
    {
        assert(_capacity > 0);
        capacity = _capacity;
    }
    int get(int key)
    {
        auto it = m.find(key);
        if(it == m.end())
            return -1;
        dll.moveToEnd(it->second.second);
        return it->second.first;
    }
    void put(int key, int value)
```

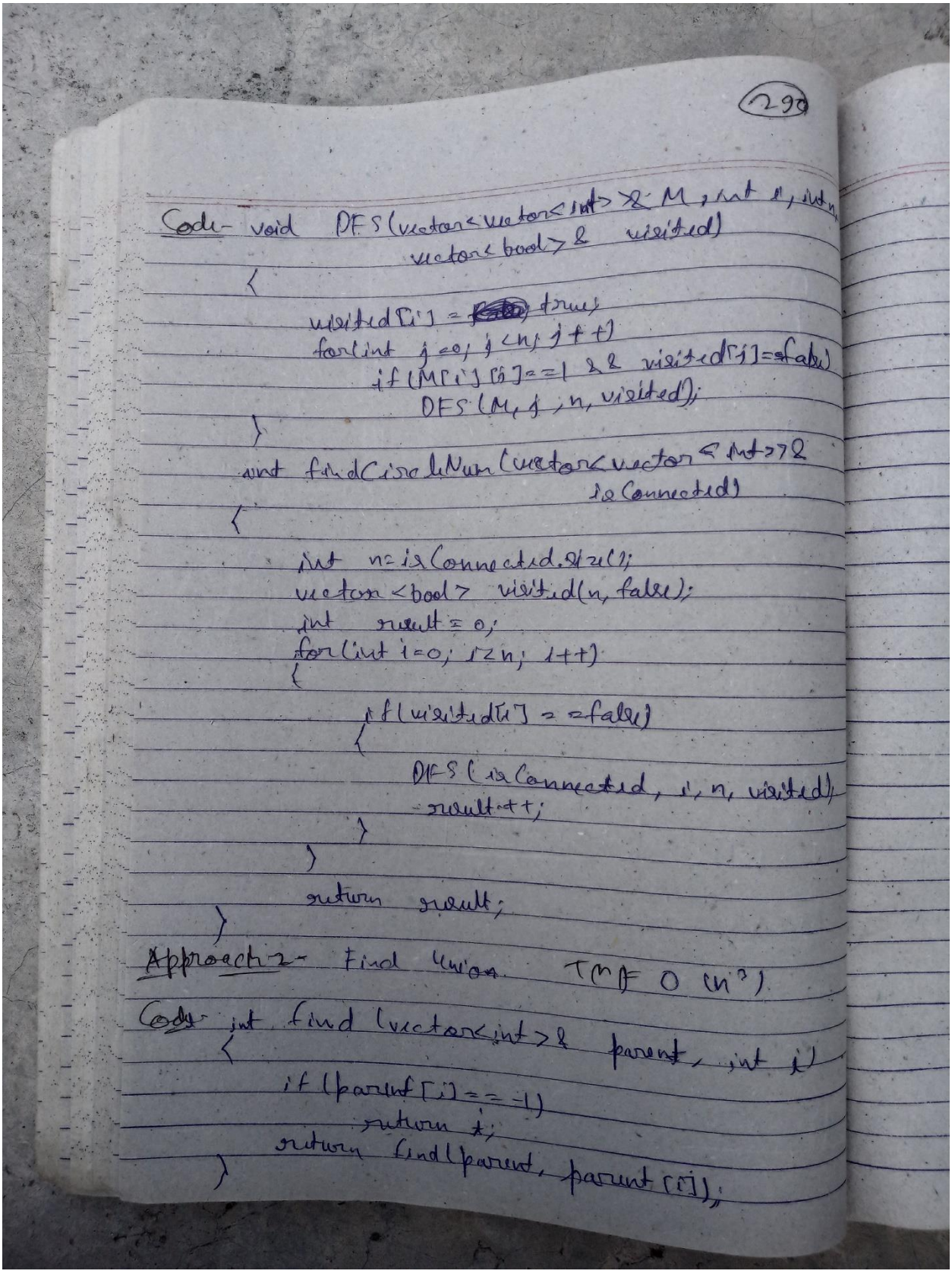


287

```
{
    auto it = m.find(key);
    if(it == m.end())
    {
        Node * n = dll.append(key);
        m[key] = {value, n};
        if(m.size() > capacity)
            m.erase(dll.deleteHead());
    }
    else
    {
        it->second.first = value;
        dll.moveToEnd(it->second.second);
    }
};
```


288
Disjoint Sets
G1
. Disjoint Sets & operations
. Detecting a cycle
. Graphical Representation
. Array Representation
. Weighted union and collapsing Find.
Operations
. Find
. Union
$S_1 \cap S_2 = \phi$
$S_1 \cup S_2 = \{1, 2, 3, 4, 5, 6, 7, 8\}$
. Cycle (10, 50)
Ex- U = {1, 2, 3, 4, 5, 6, 7, 8}
$S_1 = \{1, 2\}$
$S_2 = \{3, 4\}$
$S_1 \cup S_2 = \{1, 2, 3, 4\} = S_5$
$S_3 = \{5, 6\}$
$S_6 = \{1, 2, 3, 4, 5, 6\}$
$S_4 = \{7, 8\}$
$S_7 = \{1, 2, 3, 4, 5, 6, 7, 8\}$
Union Find Algo
Graphical Representation
U = {1, 2, 3, 4, 5, 6, 7, 8}
1 2 3 4 5 6 7 8
10
20
30
40
50
60
70
80
S_1
S_2


89
S1 = {1,2}
S2 = {3,4}
S3 = {5,6}
S4 = {7,8}
Array Representation -
Parent -
1 2 3 4 5 6 7 8
Friend Circles
O/P - 3.
Approach - Use DFS
T(n) = O(n²)

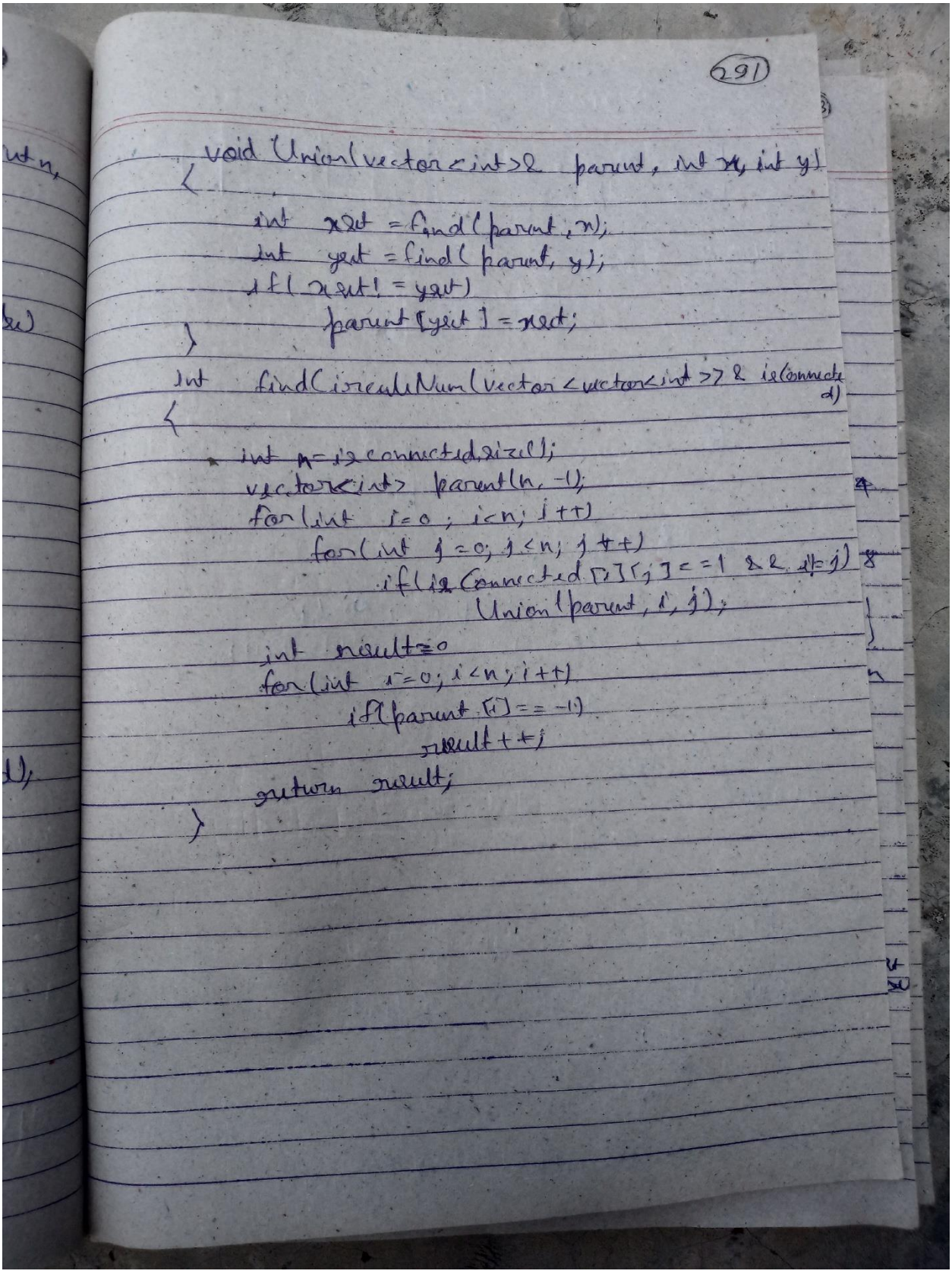


290

Code- void DFS(vector<vector<int>>& M, int i, int n,
vector<bool>& visited)
{
visited[i] = true;
for(int j=0; j<n; j++)
if(M[i][j]==1 && visited[j]==false)
DFS(M, j, n, visited);
}
int findCircleNum(vector<vector<int>>&
isConnected)
{
int n=isConnected.size();
vector<bool> visited(n, false);
int result = 0;
for(int i=0; i<n; i++)
{
if(visited[i] == false)
{
DFS(isConnected, i, n, visited);
result++;
}
}
return result;
}

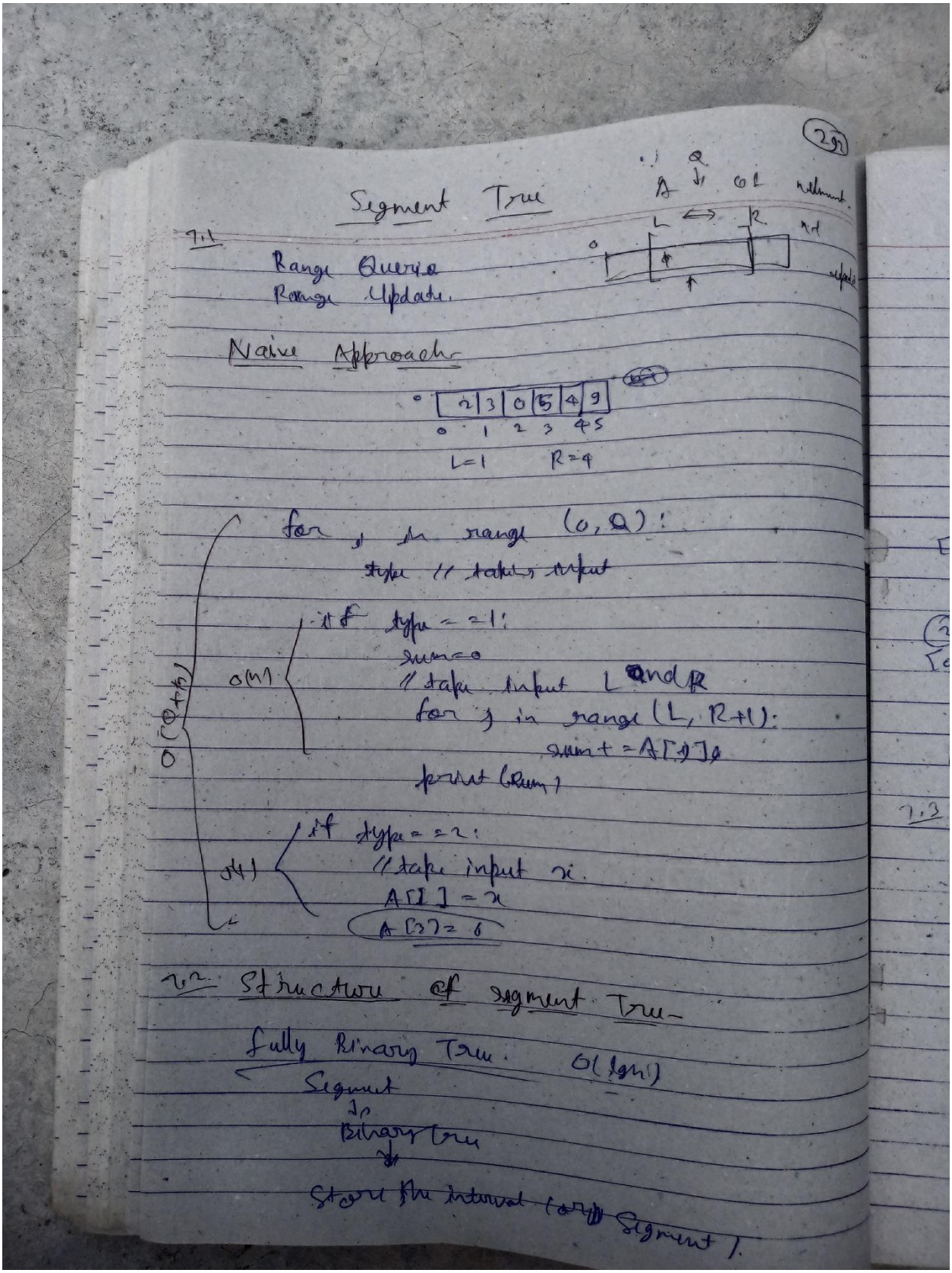
Approach 2- Find Union T(n) = O(n²)

Code- int find(vector<int>& parent, int i)
{
if(parent[i] == -1)
return i;
return find(parent, parent[i]);
}



291

```
void Union(vector<int>& parent, int x, int y)
{
    int xset = find(parent, x);
    int yset = find(parent, y);
    if(xset != yset)
        parent[yset] = xset;
}
int findCircleNum(vector<vector<int>>& isConnected)
{
    int n = isConnected.size();
    vector<int> parent(n, -1);
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            if(isConnected[i][j]==1 && i!=j)
                Union(parent, i, j);
    int result=0
    for(int i=0; i<n; i++)
        if(parent[i]==-1)
            result++;
    return result;
}
```

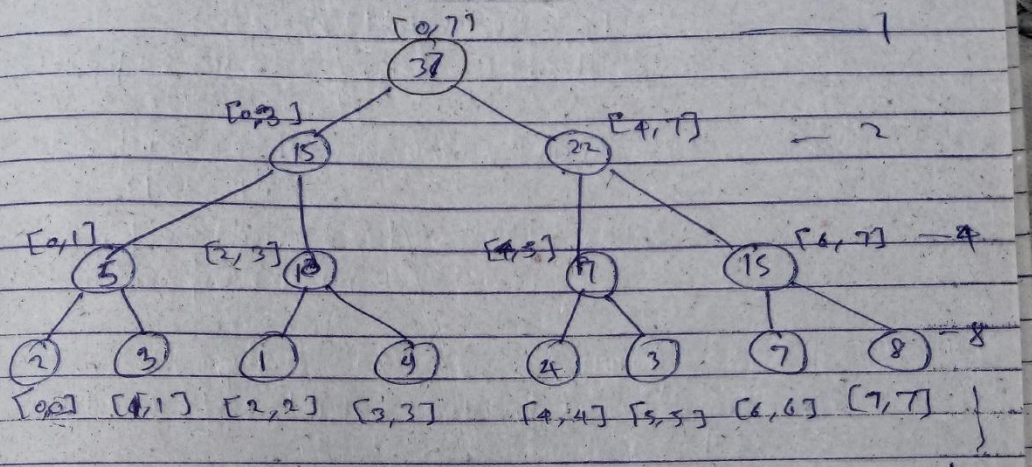

Segment Tree
Range Queries
Range Updates
Naive Approach
2 3 0 5 4 9
0 1 2 3 4 5
L=1 R=4
for i in range (0, Q):
type = int(input())
if type == 1:
sum = 0
take input L and R
for j in range (L, R+1):
sum += A[j]
print(sum)
O(N)
O(Q*N)
if type == 2:
take input x.
A[I] = x
A[3] = 6
O(1)
Structure of segment Tree-
Fully Binary Tree.
O(log N)
Segment
Binary Tree
Store the interval for each Segment.

293

0 1 2 3 4 5 6 7

A = 2, 3, 1, 9, 4, 3, 7, 8

Our task sum of all the element L to R.

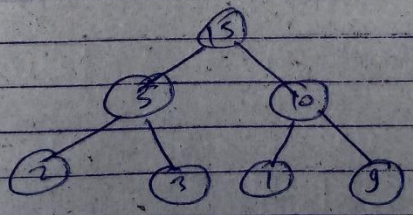


Height = O(log n)

No. of node present = 2^h

2.3 Constructing Segment Tree -

A = [2, 3, 1, 9]



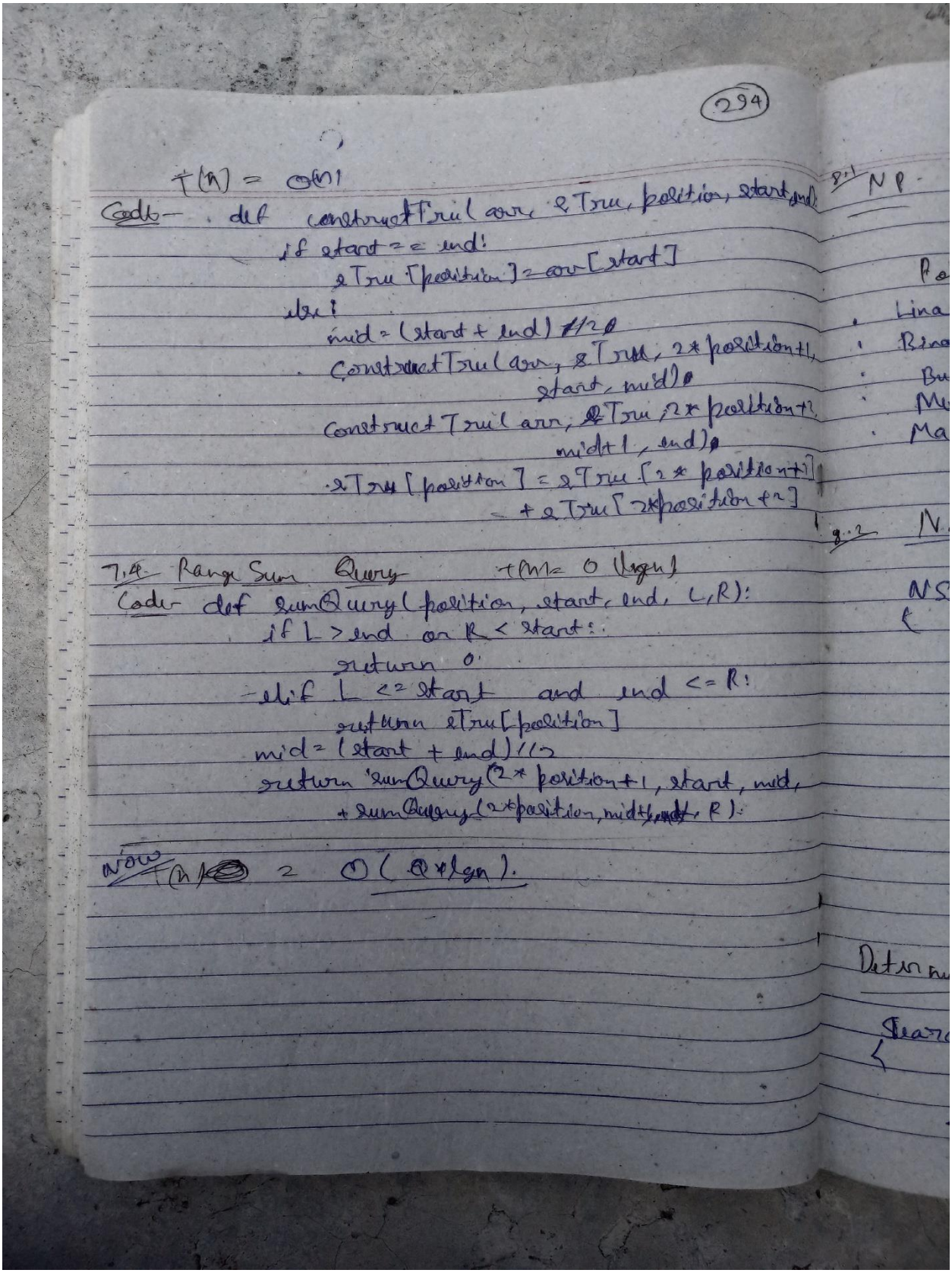
2 * 4 + 1 = 9

power of 2

filled rest nodes

2^3

$2 \times 2^3 - 1 = 15$



294

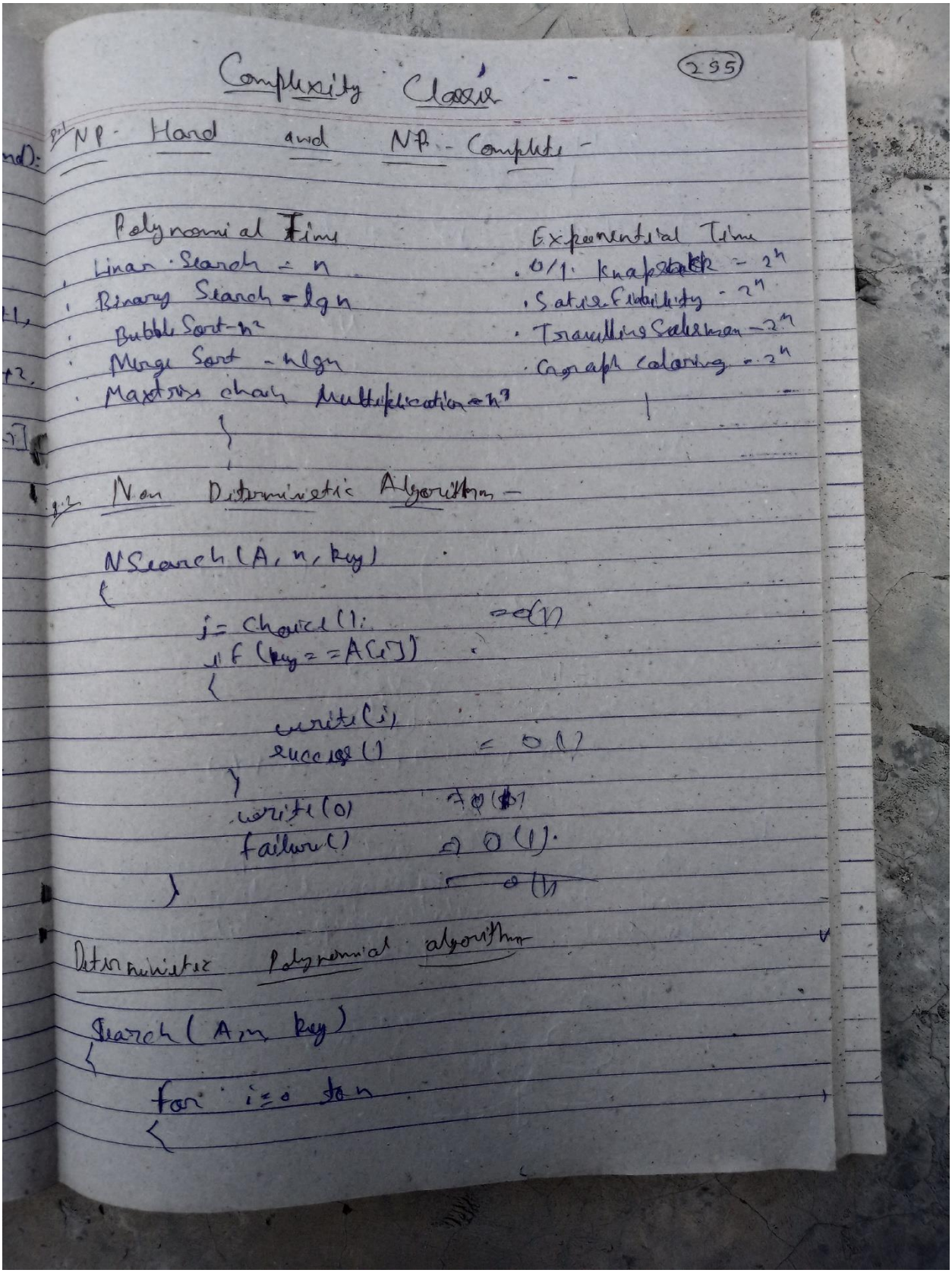
$T(n) = O(n)$

Code- def constructTree(arr, segTree, position, start, end):
if start == end:
segTree[position] = arr[start]
else:
mid = (start + end) // 2
constructTree(arr, segTree, 2*position+1, start, mid)
constructTree(arr, segTree, 2*position+2, mid+1, end)
segTree[position] = segTree[2*position+1] + segTree[2*position+2]

7.4 Range Sum Query $T(n) = O(\log n)$

Code- def sumQuery(position, start, end, L, R):
if L > end or R < start:
return 0
elif L <= start and end <= R:
return segTree[position]
mid = (start + end) // 2
return sumQuery(2*position+1, start, mid, L, R) + sumQuery(2*position+2, mid+1, end, L, R)

Now $T(n) = O(Q * \log n)$.



Complexity Classes (295)

NP - Hard and NP - Complete -

Polynomial Time
- Linear Search = n
- Binary Search = lg n
- Bubble Sort = n^2
- Merge Sort = n lg n
- Matrix chain Multiplication = n^3

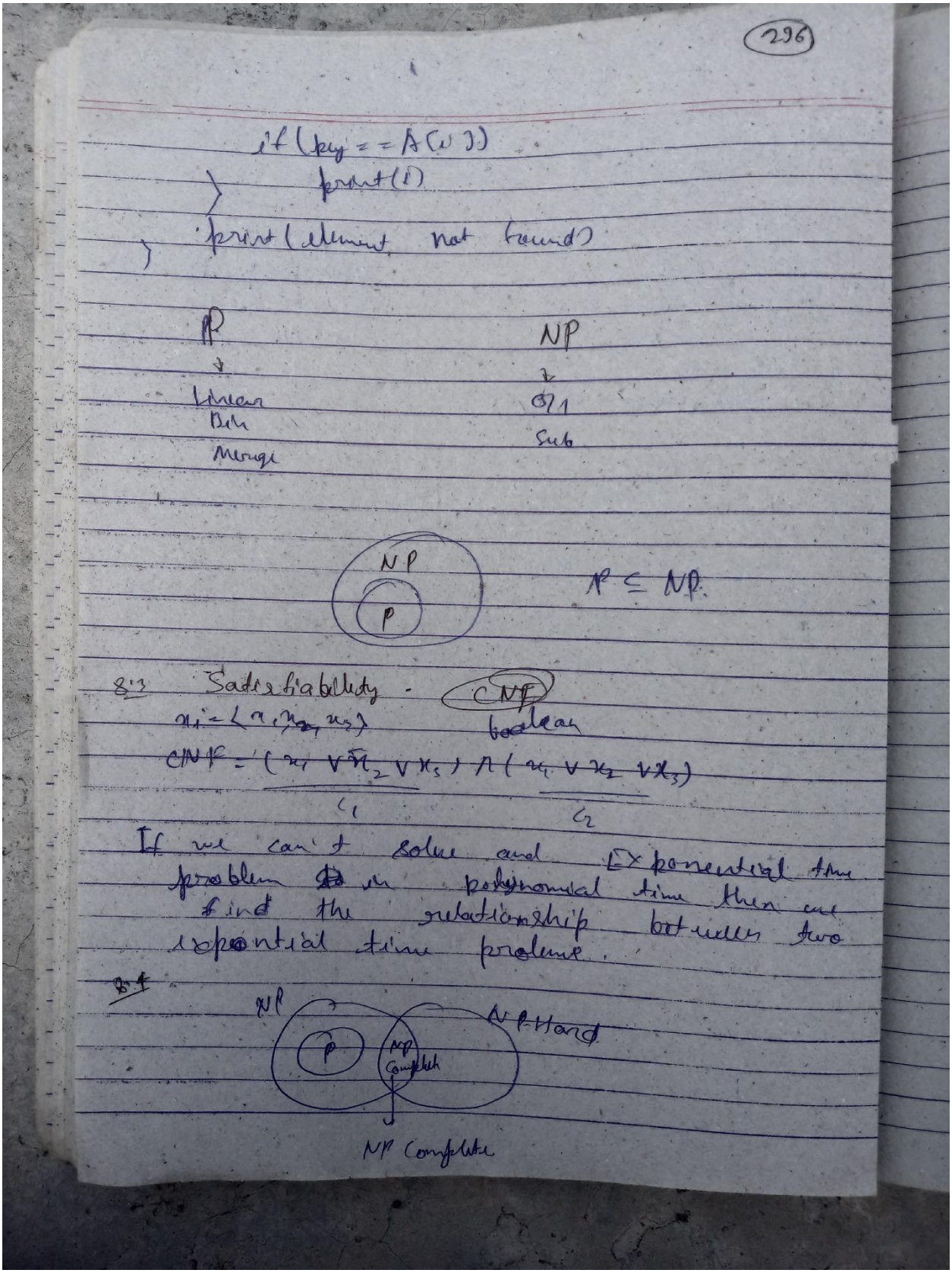
Exponential Time
- 0/1 knapsack = 2^n
- Satisfiability = 2^n
- Travelling Salesman = 2^n
- Graph coloring = 2^n

Non Deterministic Algorithm -

```
NSearch (A, n, key)
{
    i = Choice (1:n)        -> O(n)
    if (key == A[i])
    {
        write (i)
        success ()          -> O(1)
    }
    write (0)               -> O(1)
    failure ()              -> O(1).
}                           -> O(n)
```

Deterministic Polynomial algorithm

```
Search (A, n, key)
{
    for i = 0 to n
    {
```


296
if (key == A[i])
print(i)
print(element not found)
P
Linear
Bin
Merge
NP
QT
Sub
NP
P
P ⊆ NP.
8.3 Satisfiability -
CNF
boolean
CNF = (x1 ∨ x̄2 ∨ x3) ∧ (x1 ∨ x2 ∨ x3)
C1
C2
If we can't solve and Exponential time problem in polynomial time then we find the relationship between two exponential time problems.
8.4
NP
NP-Hard
P
NP Complete
NP Complete