Design and Analysis of Algorithms (CS206)

LAB Assignment – 5

1.1. (T) Find a computational problem that you can solve using the divide and conquer approach. The problem should be different than the problems discussed in class, and it should be **unique** and **interesting**.

**HELP BOB!**

Bob, the Builder is working in Skyline Real Estate Company, and his Company has assigned him Project 'DEMOLITION'.

In Project DEMOLITION, Skyline Real Estate Developers is planning to demolish a number of old, unoccupied buildings and construct a shopping mall in their place.

Bob's task is to find the largest solid area in which the mall can be constructed.

There are a number of buildings in a certain two-dimensional landscape.

Each building has a height, given by h[i] where i belongs [1, n].

If you Join k adjacent buildings, they will form solid rectangle of area:



But, Bob would get Promotion, if the Area Demolished is Maximum!

Would you Help Bob to solve this Problem to get Maximum Area, he can demolished?

**Input Format**

Input File Containing space-separated integers, each representing the height of a building.

**Constraints**



**Output Format**

Print long integer, representing the maximum area of rectangle that can be formed.

Remember that this Rectangle must be aligned at common base line.

For Example,

*Sample Input:*

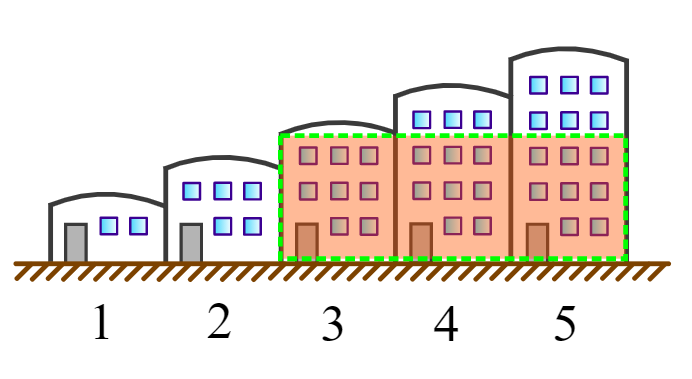


*Sample Output:*



*Explanation:*

Maximum Area {3,4,5} = 3\*[3] = 9

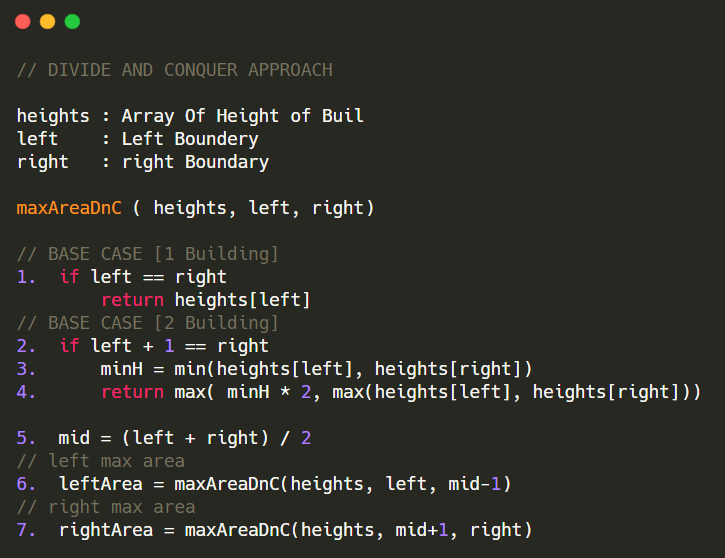


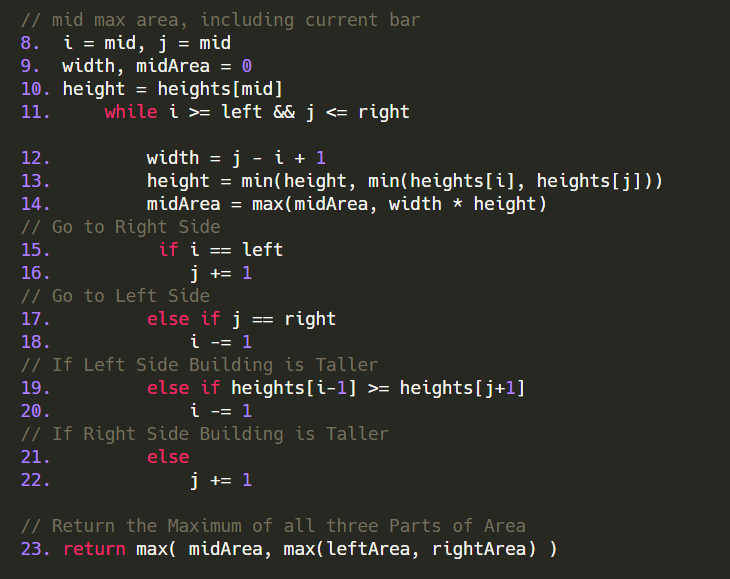
1.2. (T) Write pseudocodes to design algorithms for the above mentioned computational problem using the brute-force approach (incremental approach) and the divide and conquer approach.

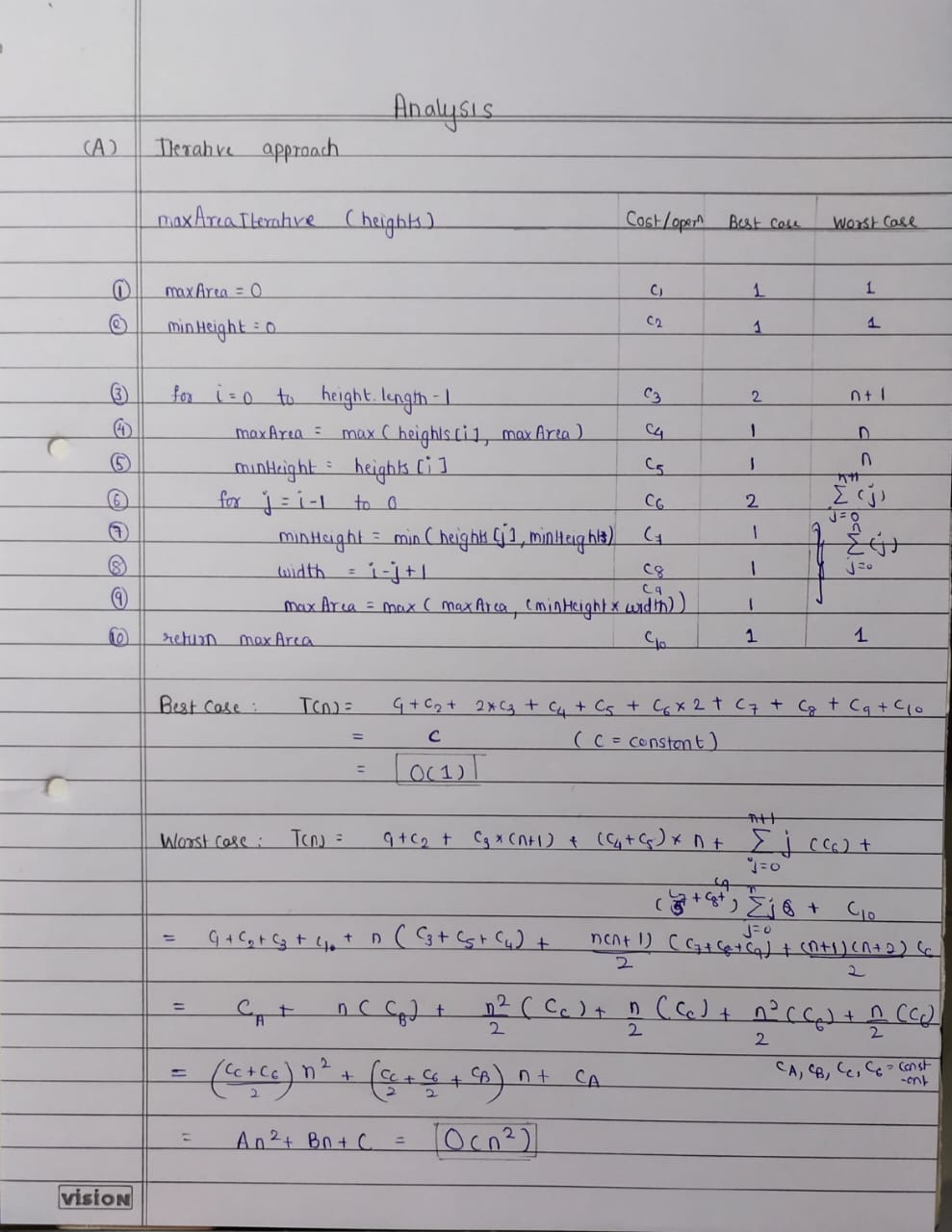
**Pseudo-Code INCREMENTAL**

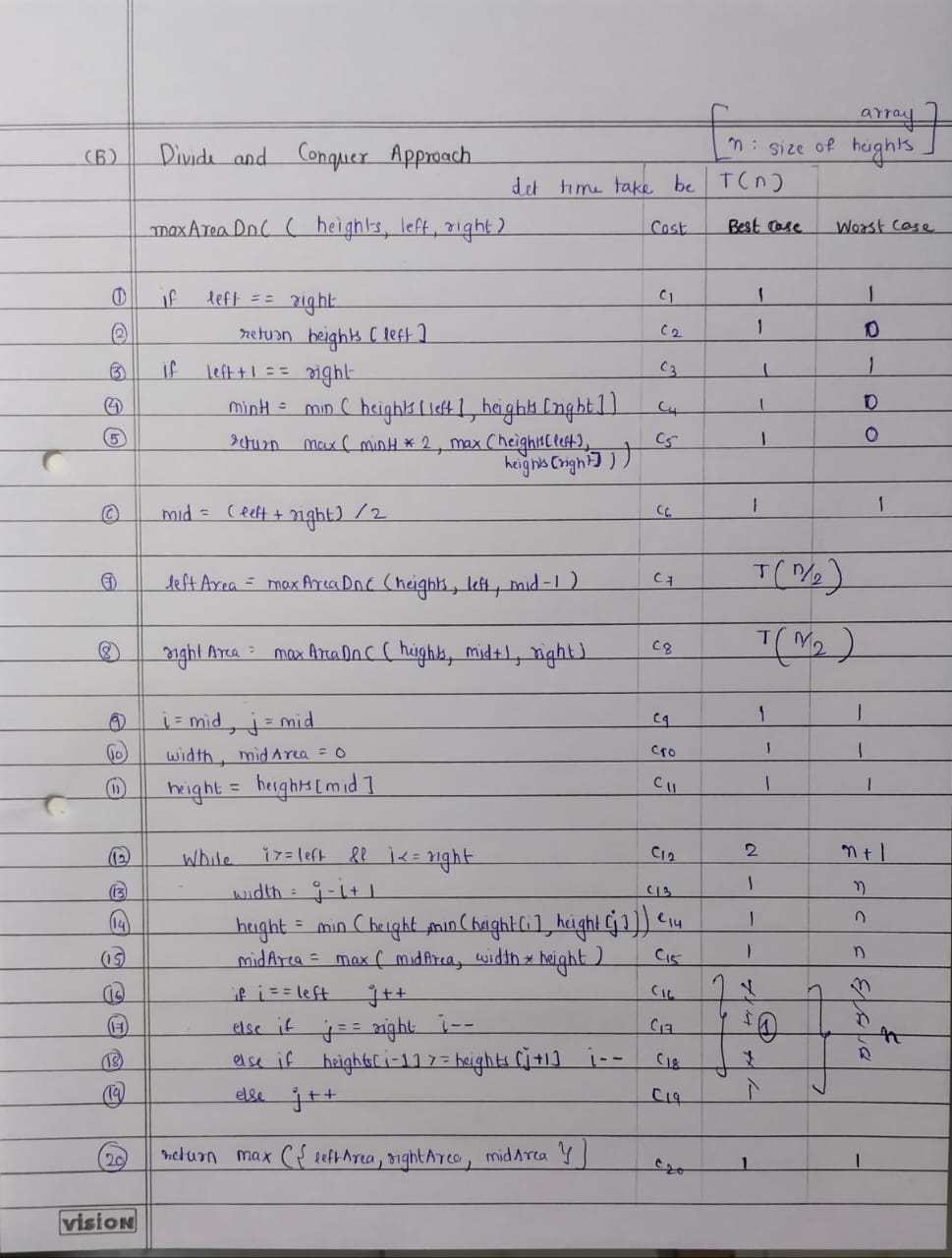


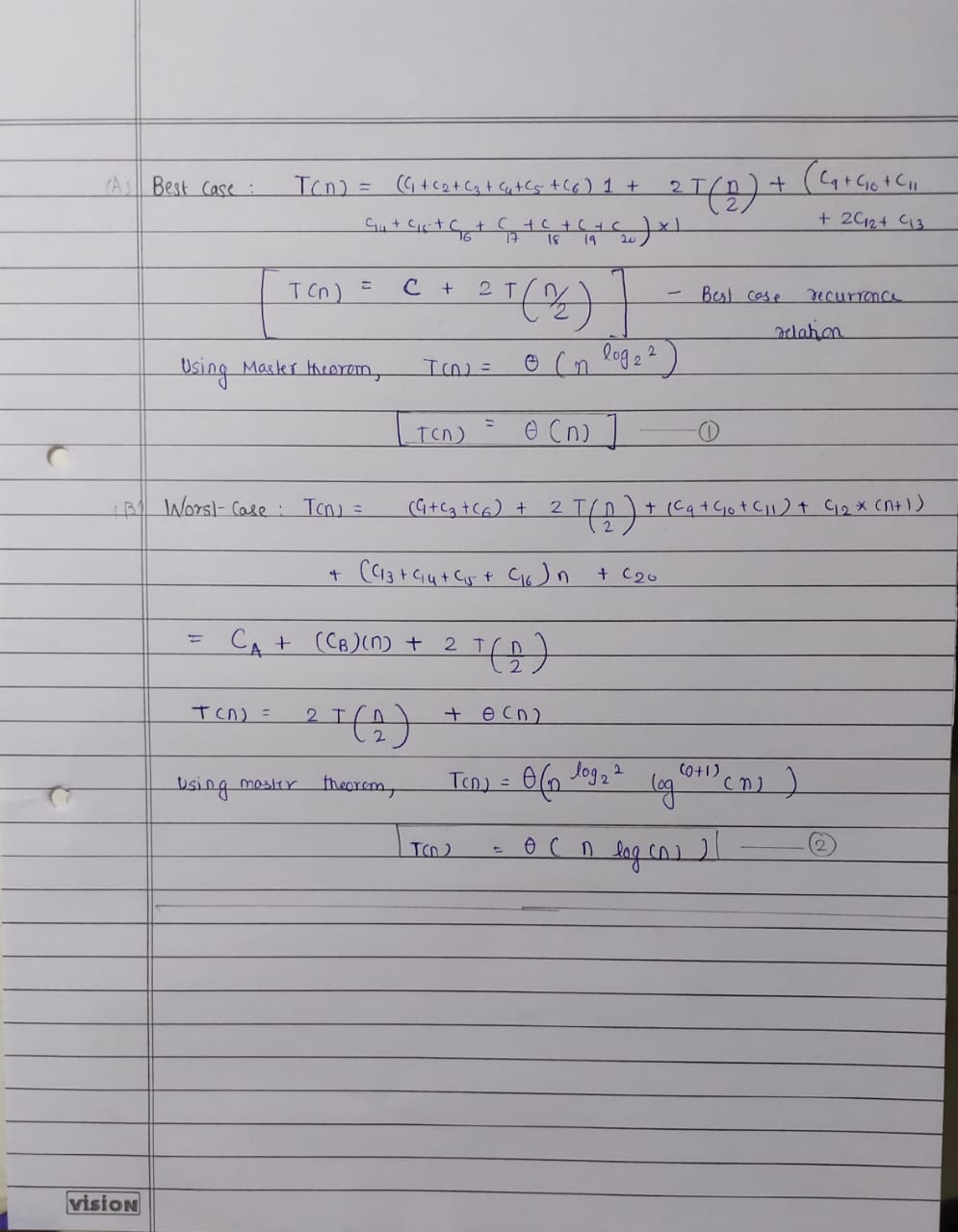
**Pseudo-Code DIVIDE AND CONQUER**







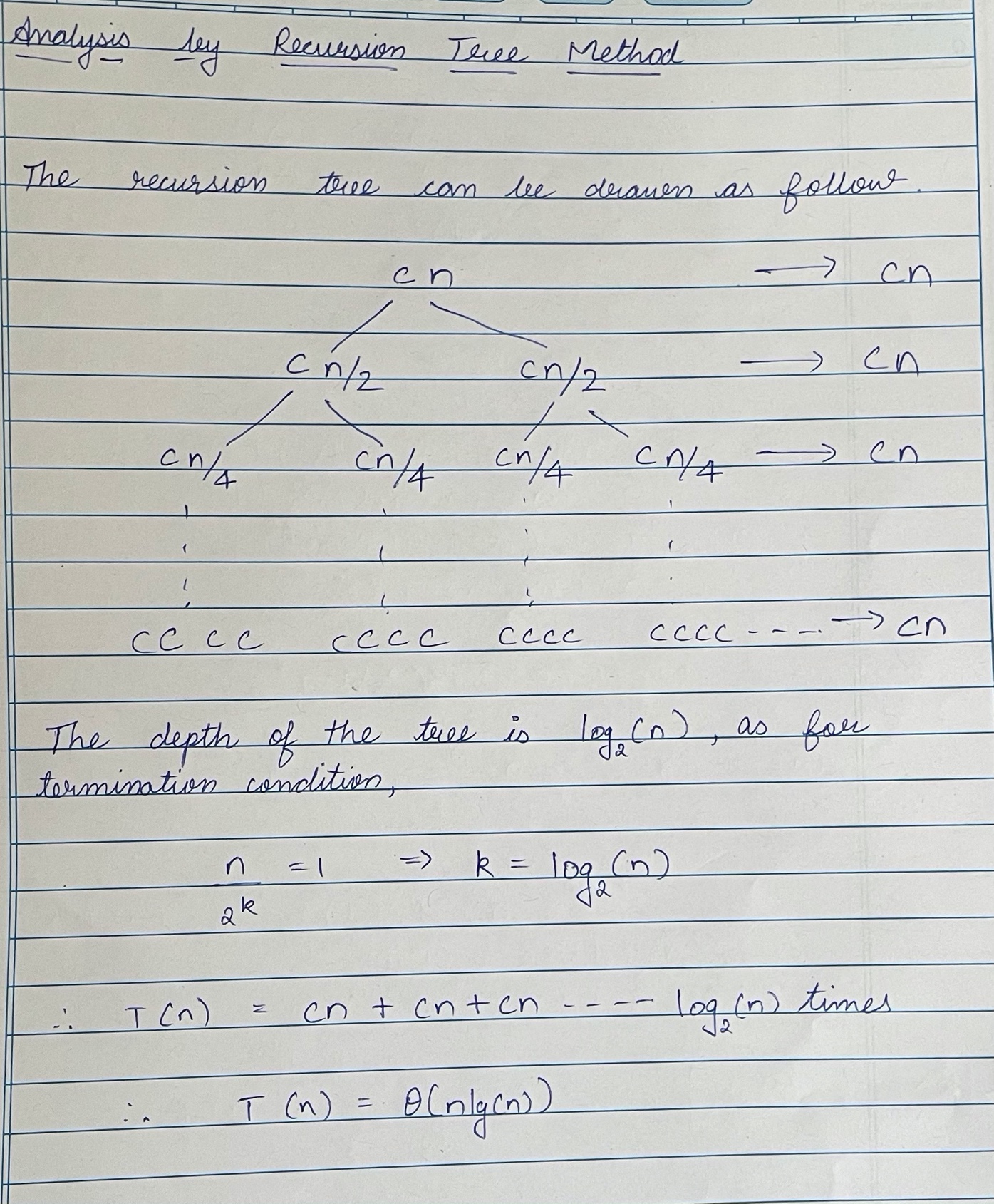




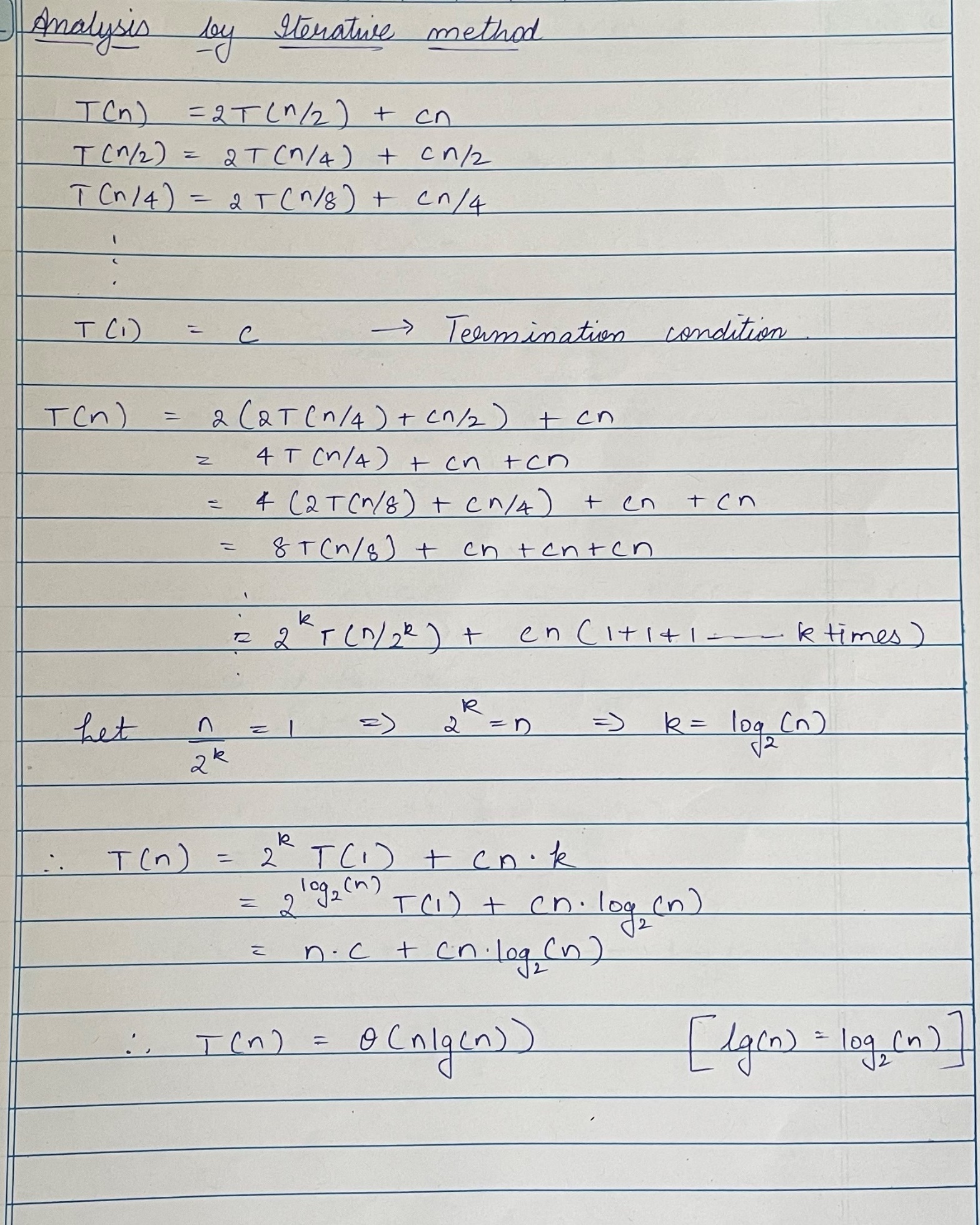
1.3. (T) Analyze the time complexity of above algorithms.

Analyze the divide and conquer algorithm using different methods such as:

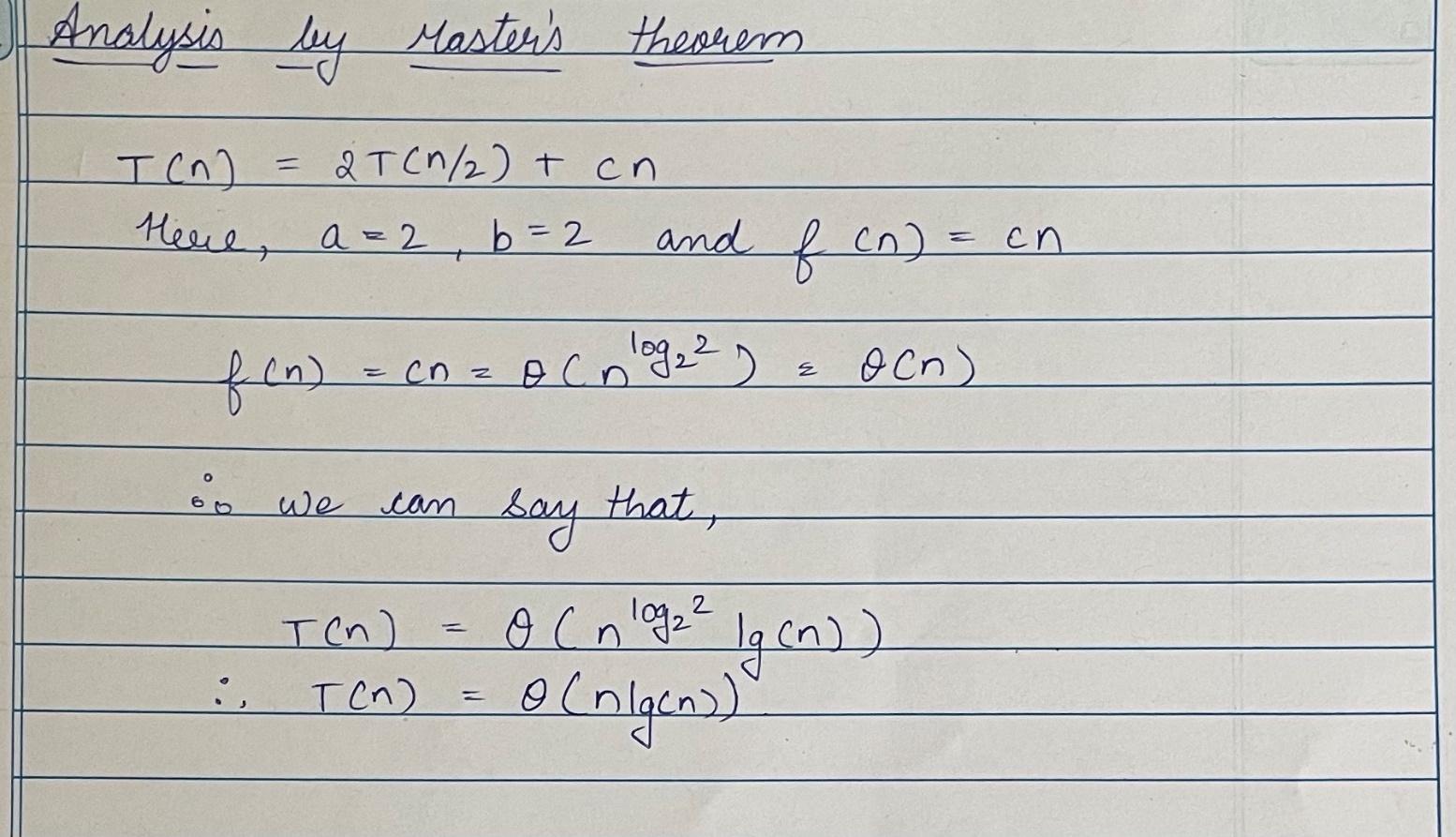
**(1) Recursion Tree Method**



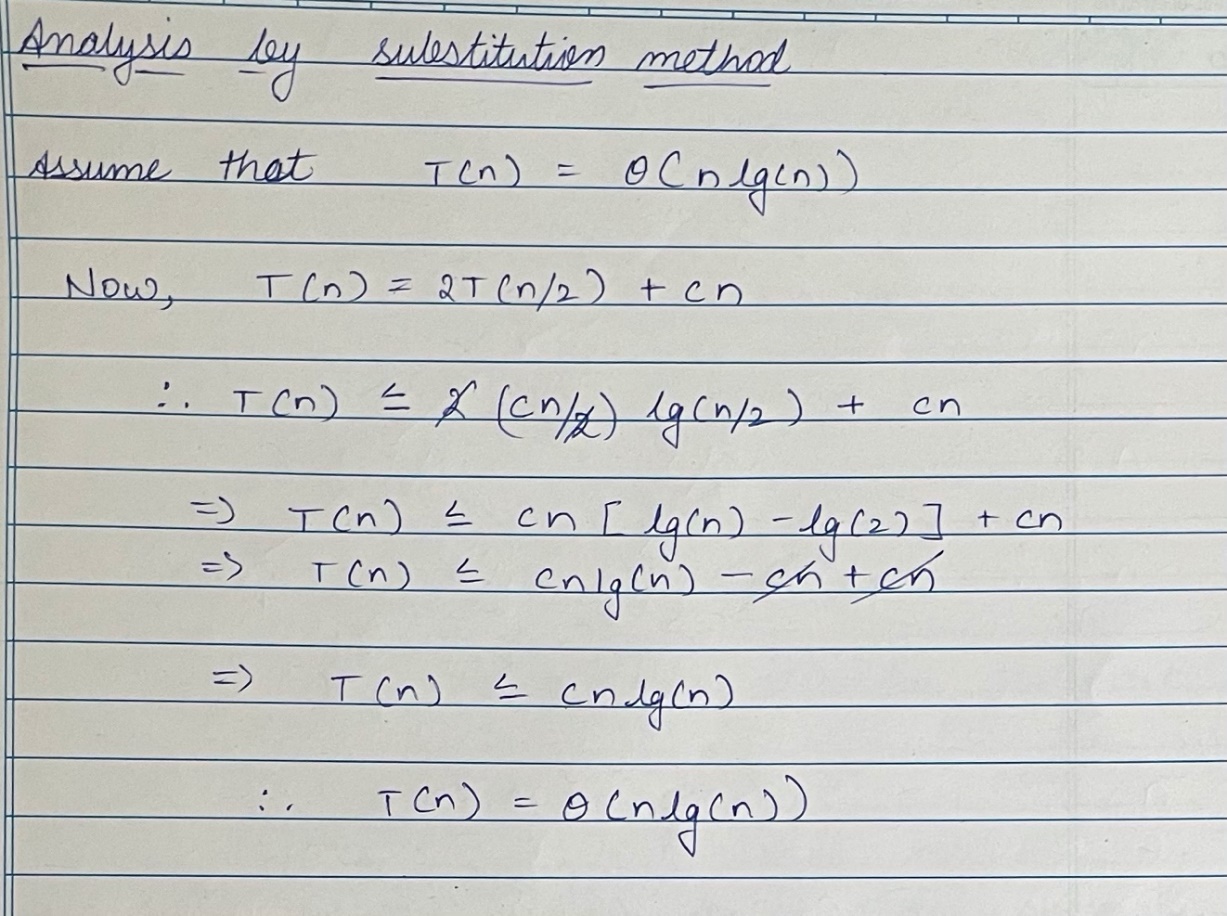
**(2) Iterative Method**



**(3) Master Method**



**(4) Substitution Method.**



1.4. (L) Provide the details of Hardware/Software you used to implement algorithms and to measure the time.

Hardware Details of Laptop used for testing:

|  |  |
| --- | --- |
| PARAMETER | LAPTOP CONFIGURATION |
| Operating System | Microsoft Windows **10 v-**20H2 |
| Processor | Intel(R) Core(TM) i5-9300H [Core **i5 9th Gen**] |
| CPU | **2.30GHz(base)**, **4GHz(boost)**, **4** Core(s), 8 Logical Processor(s) |
| System Type | x64-based PC [**64 Bit**] |
| RAM | **8.00** GB |
| Hard Drive/SSD | 512 GB **SSD** |

Software Used:

|  |  |
| --- | --- |
| PARAMETER | LAPTOP CONFIGURATION |
| Code Editor | **Intellij IDEA Community Edition 2020.3.1** |
| Compiler | jdk-13.0.2 |
| Time | Measured using **System.nanotime()** |
| Programming Language Used | **Java** |

1.5. (L) Implement the above algorithms and submit the code (complete programs).

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Scanner;

public class maxAreaHistogram {

    public static void main(String[] args) {

*try* {

            File output = *new* File("output.txt");

            output.createNewFile();

            FileWriter writer = *new* FileWriter(output);

            ArrayList<Integer> l = *new* ArrayList<>();

            long startTime, endTime, timeTaken;

            String file;

*for* (int i = 1; i < 4; i++) {

                file = String.format("File %d.txt", i);

*try* {

                    loadFile(l, file);

                } *catch* (FileNotFoundException e) {

                    e.printStackTrace();

                }

                int[] heights = l.stream()

                .mapToInt(Integer*::*intValue)

                .toArray();

                int size = heights.length;

                int ans = 0;

                writer.write("File " + i + "\n");

                System.out.println("File " + i);

*// Iterative*

                timeTaken = 0;

                startTime = System.nanoTime();

*for* (int j = 0; j < 20; j++) {

                    ans = maxAreaIterative(heights);

                }

                endTime = System.nanoTime();

                timeTaken = (endTime - startTime);

                writer.write(ans + ", ");

                System.out.print(ans + ", ");

                writer.write(timeTaken / 20 + " ns, ");

                System.out.print(timeTaken / 20 + " ns, ");

*// Divide and conquer*

                timeTaken = 0;

                startTime = System.nanoTime();

*for* (int j = 0; j < 20; j++) {

                    ans = maxAreaDnC(heights, 0, size - 1);

                }

                endTime = System.nanoTime();

                timeTaken = (endTime - startTime);

                writer.write(ans + ", ");

                System.out.print(ans + ", ");

                writer.write(timeTaken / 20 + " ns\n");

                System.out.println(timeTaken / 20 + " ns");

                l.clear();

            }

            writer.close();

        } *catch* (IOException e) {

            e.printStackTrace();

        }

    }

    public static void loadFile(ArrayList<Integer> A, String file) throws FileNotFoundException {

        Scanner s = *new* Scanner(*new* File(file));

        int temp;

*while* (s.hasNext()) {

            temp = s.nextInt();

            A.add(temp);

        }

    }

    public static int maxAreaIterative(int[] heights) {

        int maxArea = 0;

        int minHeight = 0;

*for* (int i = 0; i < heights.length; i++) {

            maxArea = Math.max(heights[i], maxArea);

            minHeight = heights[i];

*for*(int j = i - 1; j >= 0 ; j--) {

                minHeight = Math.min(heights[j], minHeight);

                int width = (i - j + 1);

                maxArea = Math.max(maxArea, (minHeight \* width));

            }

        }

*return* maxArea;

    }

    public static int maxAreaDnC(int[] heights, int left, int right) {

*if* (left == right) {

*return* heights[left];

        }

*if* (left + 1 == right) {

            int minH = Math.min(heights[left], heights[right]);

*return* Math.max(minH \* 2, Math.max(heights[left], heights[right]));

        }

        int mid = (left + right) / 2;

*// left max area*

        int leftArea = maxAreaDnC(heights, left, mid-1);

*// right max area*

        int rightArea = maxAreaDnC(heights, mid+1, right);

*// mid max area, including current bar*

        int i = mid, j = mid;

        int width, midArea = 0;

        int height = heights[mid];

*while* (i >= left && j <= right) {

            width = j - i + 1;

            height = Math.min(height, Math.min(heights[i], heights[j]));

            midArea = Math.max(midArea, width \* height);

*if* (i == left) {

                j += 1;

            } *else* *if* (j == right) {

                i -= 1;

            } *else* *if* (heights[i-1] >= heights[j+1]) {

                i -= 1;

            } *else* {

                j += 1;

            }

        }

*return* Math.max( midArea, Math.max(leftArea, rightArea) );

    }

}

1.6. (L) Analyze the performance of both the implemented algorithms (performance of algorithms on your computers). Plot a graph.

BRUTE FORCE (ITERATIVE)

|  |  |  |
| --- | --- | --- |
| Sr. No. | No of Elements | Brute Force (ms) |
| 1 | 1024 | 21.51845 |
| 2 | 2048 | 96.03287 |
| 3 | 4096 | 424.20708 |
| 4 | 8192 | 1539.63372 |
| 5 | 16384 | 5947.25934 |
| 6 | 32768 | 29003.11334 |
| 7 | 65536 | 110500.4987 |
| 8 | 131072 | 406734.3615 |
| 9 | 262144 | 1636508.868 |

* Here T(n) = 0.0000236585\*n2 + 0.0359048\*n + 473.34

DIVIDE AND CONQUER

|  |  |  |
| --- | --- | --- |
| Sr. No. | No of Elements | Divide and Conquer (ms) |
| 1 | 1024 | 0.88949 |
| 2 | 2048 | 1.66137 |
| 3 | 4096 | 3.6958 |
| 4 | 8192 | 8.0214 |
| 5 | 16384 | 17.09201 |
| 6 | 32768 | 39.39826 |
| 7 | 65536 | 76.16316 |
| 8 | 131072 | 161.02624 |
| 9 | 262144 | 354.04978 |

* Here T(n) = 0.00010009\*n\*log2n - 0.000459491\*n + 1.2401

1.7. (L) Comparatively Analyze the performance of above algorithms and plot a graph.

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | No of Elements | Brute Force (ms) | Divide and Conquer  (ms) |
| 1 | 1024 | 21.51845 | 0.88949 |
| 2 | 2048 | 96.03287 | 1.66137 |
| 3 | 4096 | 424.20708 | 3.6958 |
| 4 | 8192 | 1539.63372 | 8.0214 |
| 5 | 16384 | 5947.25934 | 17.09201 |
| 6 | 32768 | 29003.11334 | 39.39826 |
| 7 | 65536 | 110500.4987 | 76.16316 |
| 8 | 131072 | 406734.3615 | 161.02624 |
| 9 | 262144 | 1636508.868 | 354.04978 |

SUBMITTED BY:

|  |  |
| --- | --- |
| Sr. No. | Admission No. |
| 1 | U19CS011 |
| 2 | U19CS012 |
| 3 | U19CS049 |
| 4 | U19CS080 |