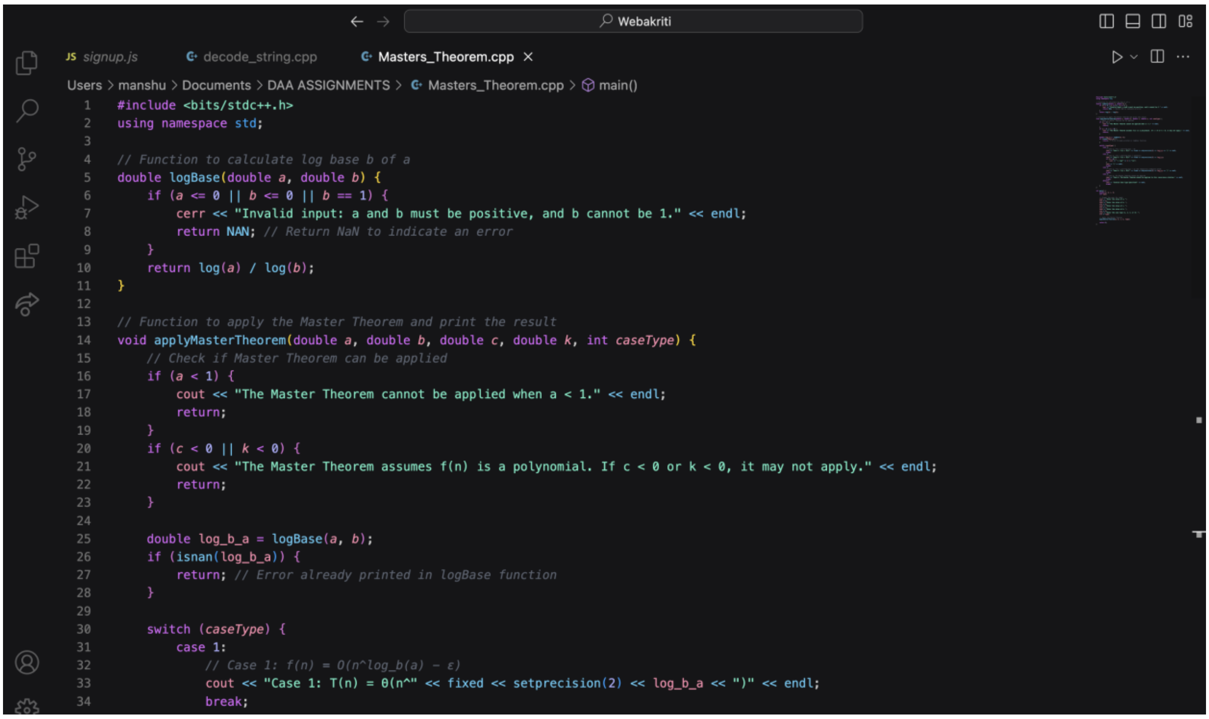
DAA ASSIGNMENT 3

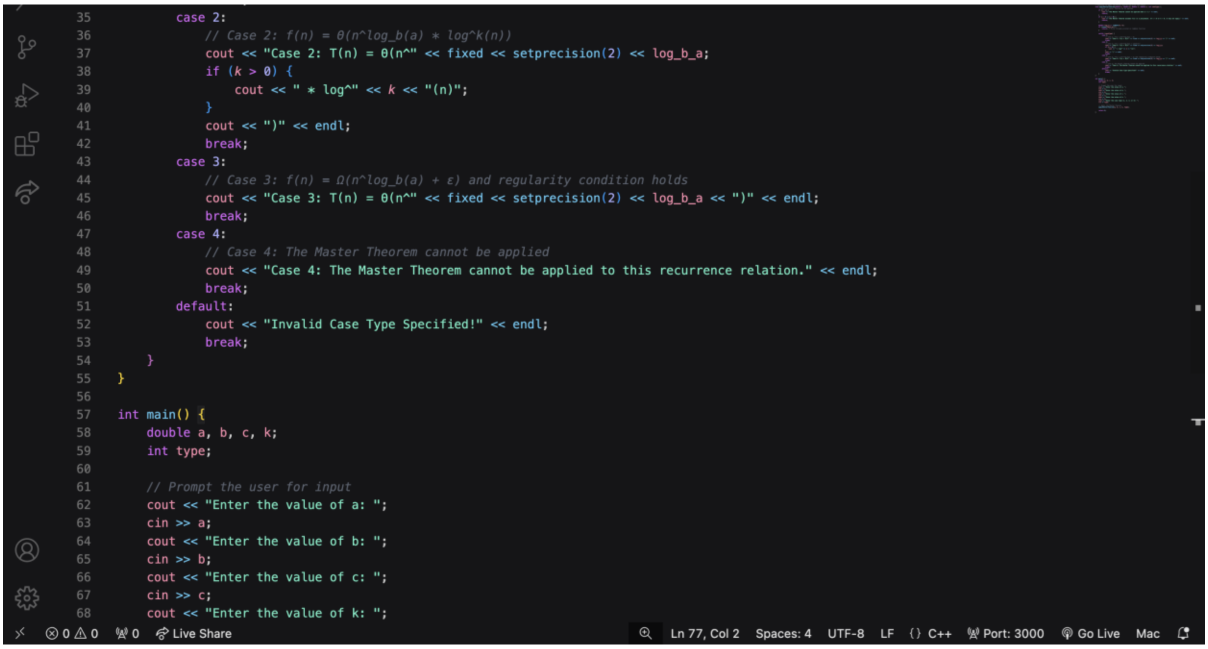
Name : Manshu Jaiswal

Roll No : 2023IMT-051

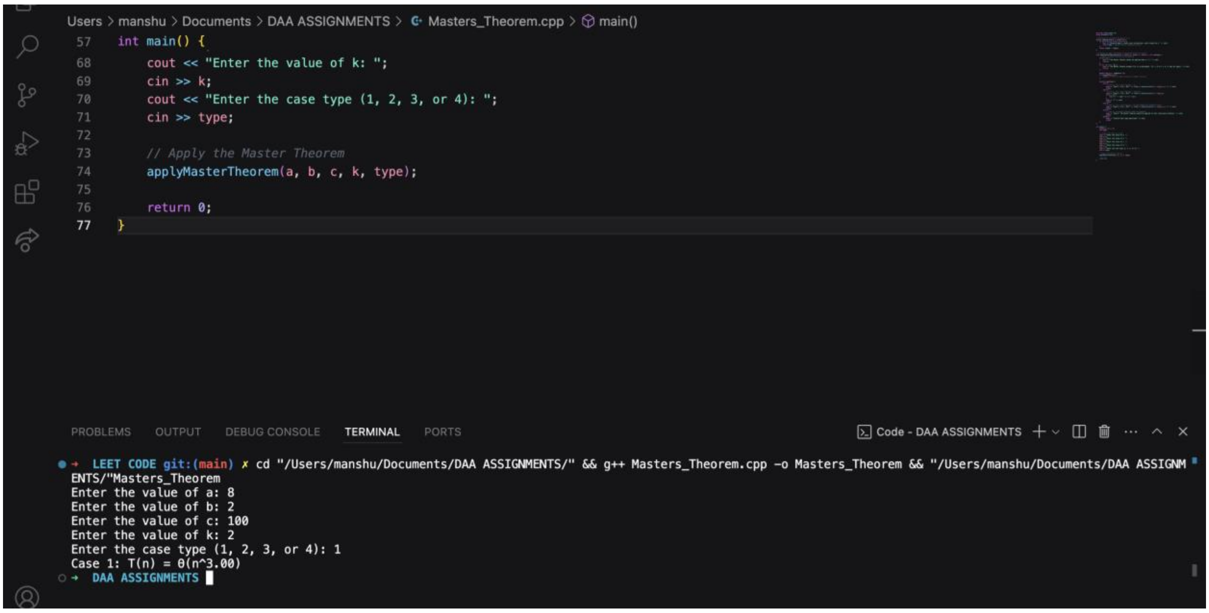
Question 1 :


Webakriti
JS signup.js
decode_string.cpp
Masters_Theorem.cpp
Users > manshu > Documents > DAA ASSIGNMENTS > Masters_Theorem.cpp > main()

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to calculate log base b of a
double logBase(double a, double b) {
    if (a <= 0 || b <= 0 || b == 1) {
        cerr << "Invalid input: a and b must be positive, and b cannot be 1." << endl;
        return NAN; // Return NaN to indicate an error
    }
    return log(a) / log(b);
}

// Function to apply the Master Theorem and print the result
void applyMasterTheorem(double a, double b, double c, double k, int caseType) {
    // Check if Master Theorem can be applied
    if (a < 1) {
        cout << "The Master Theorem cannot be applied when a < 1." << endl;
        return;
    }
    if (c < 0 || k < 0) {
        cout << "The Master Theorem assumes f(n) is a polynomial. If c < 0 or k < 0, it may not apply." << endl;
        return;
    }

    double log_b_a = logBase(a, b);
    if (isnan(log_b_a)) {
        return; // Error already printed in logBase function
    }

    switch (caseType) {
        case 1:
            // Case 1: f(n) = O(n^log_b(a) − ε)
            cout << "Case 1: T(n) = θ(n^" << fixed << setprecision(2) << log_b_a << ")" << endl;
            break;
```



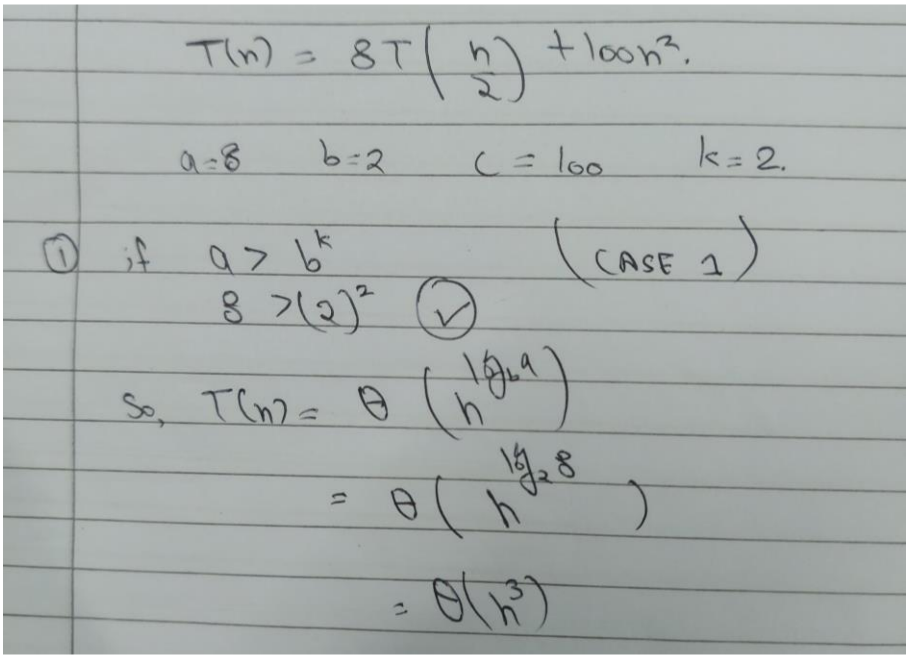
```cpp
        case 2:
            // Case 2: f(n) = θ(n^log_b(a) * log^k(n))
            cout << "Case 2: T(n) = θ(n^" << fixed << setprecision(2) << log_b_a;
            if (k > 0) {
                cout << " * log^" << k << "(n)";
            }
            cout << ")" << endl;
            break;
        case 3:
            // Case 3: f(n) = Ω(n^log_b(a) + ε) and regularity condition holds
            cout << "Case 3: T(n) = θ(n^" << fixed << setprecision(2) << log_b_a << ")" << endl;
            break;
        case 4:
            // Case 4: The Master Theorem cannot be applied
            cout << "Case 4: The Master Theorem cannot be applied to this recurrence relation." << endl;
            break;
        default:
            cout << "Invalid Case Type Specified!" << endl;
            break;
    }
}

int main() {
    double a, b, c, k;
    int type;

    // Prompt the user for input
    cout << "Enter the value of a: ";
    cin >> a;
    cout << "Enter the value of b: ";
    cin >> b;
    cout << "Enter the value of c: ";
    cin >> c;
    cout << "Enter the value of k: ";
```


Users > manshu > Documents > DAA ASSIGNMENTS > Masters_Theorem.cpp > main()
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Code - DAA ASSIGNMENTS

```cpp
int main() {
    cout << "Enter the value of k: ";
    cin >> k;
    cout << "Enter the case type (1, 2, 3, or 4): ";
    cin >> type;

    // Apply the Master Theorem
    applyMasterTheorem(a, b, c, k, type);

    return 0;
}
```

```
➜  LEET CODE git:(main) ✗ cd "/Users/manshu/Documents/DAA ASSIGNMENTS/" && g++ Masters_Theorem.cpp -o Masters_Theorem && "/Users/manshu/Documents/DAA ASSIGNM
ENTS/"Masters_Theorem
Enter the value of a: 8
Enter the value of b: 2
Enter the value of c: 100
Enter the value of k: 2
Enter the case type (1, 2, 3, or 4): 1
Case 1: T(n) = θ(n^3.00)
➜  DAA ASSIGNMENTS
```

Verification :



$$T(n) = 8T\left(\frac{n}{2}\right) + 100n^2.$$

$a=8 \quad b=2 \quad c=100 \quad k=2.$

① if $a > b^k$ (CASE 1)

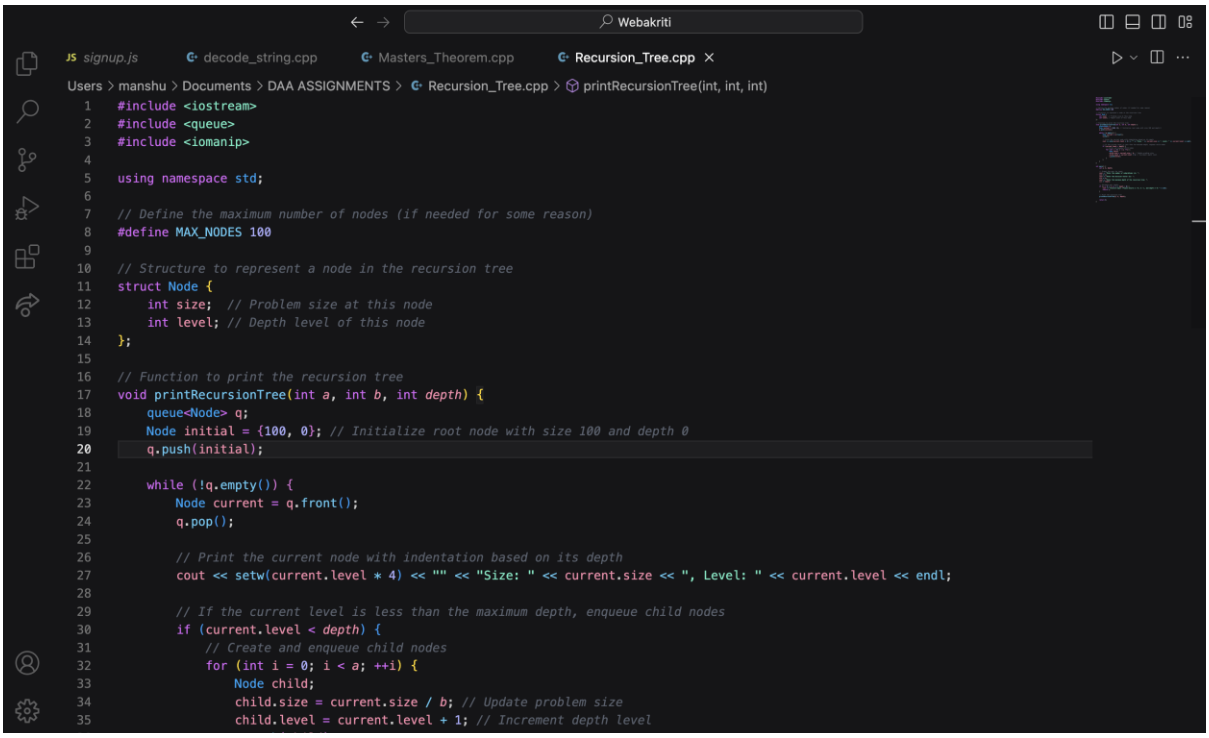
$8 > (2)^2$ ✓

So, $T(n) = \Theta\left(n^{\log_b a}\right)$

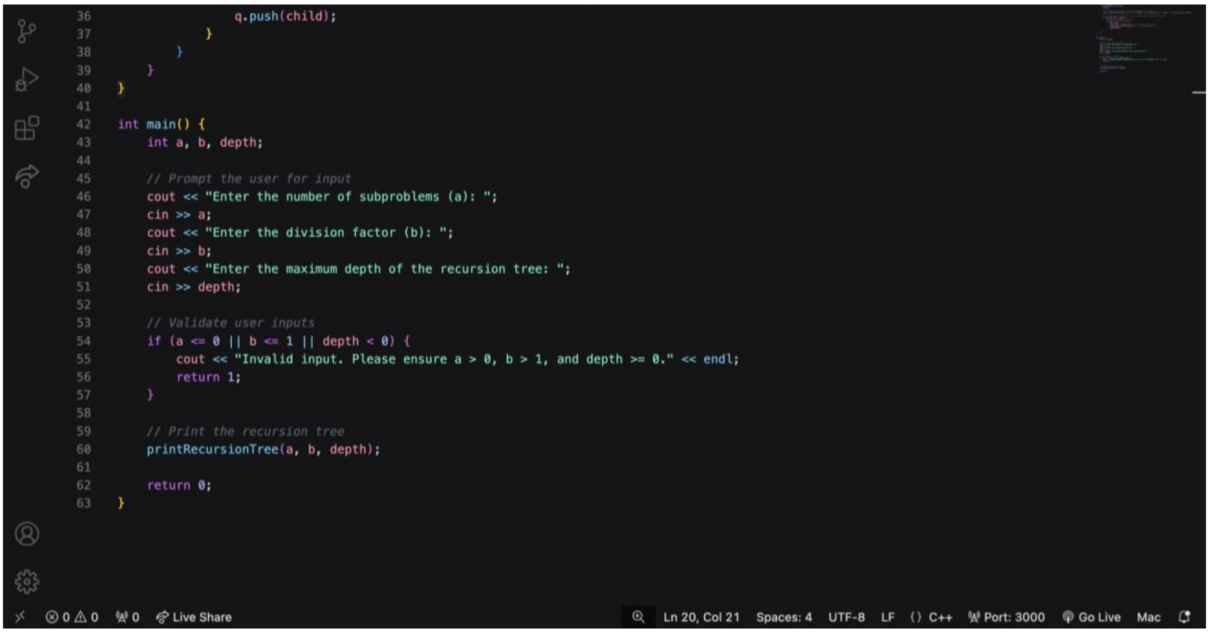
$= \Theta\left(n^{\log_2 8}\right)$

$= \Theta(n^3)$

Question 2 :

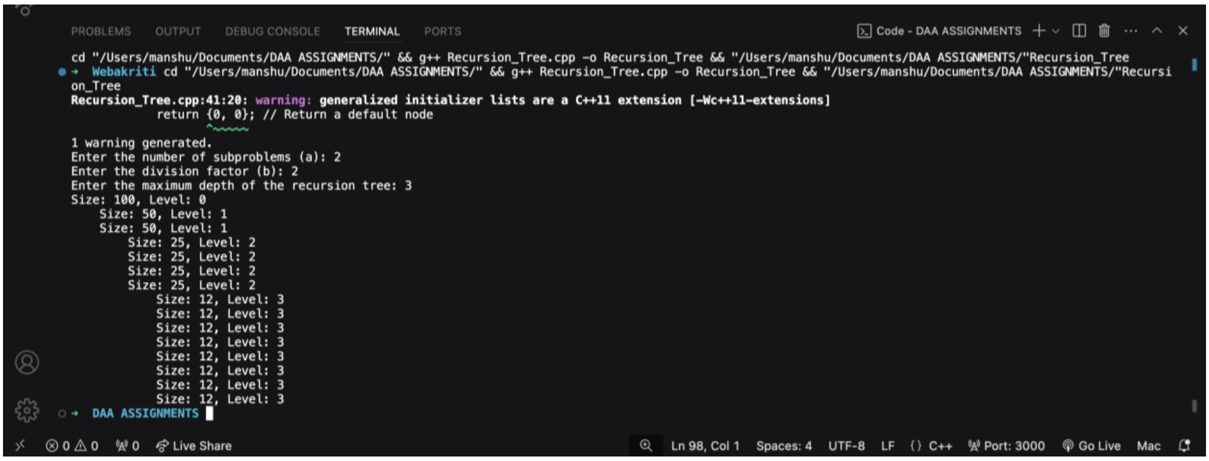

Webakriti
signup.js
decode_string.cpp
Masters_Theorem.cpp
Recursion_Tree.cpp
Users > manshu > Documents > DAA ASSIGNMENTS > Recursion_Tree.cpp > printRecursionTree(int, int, int)

```cpp
#include <iostream>
#include <queue>
#include <iomanip>

using namespace std;

// Define the maximum number of nodes (if needed for some reason)
#define MAX_NODES 100

// Structure to represent a node in the recursion tree
struct Node {
    int size;  // Problem size at this node
    int level; // Depth level of this node
};

// Function to print the recursion tree
void printRecursionTree(int a, int b, int depth) {
    queue<Node> q;
    Node initial = {100, 0}; // Initialize root node with size 100 and depth 0
    q.push(initial);

    while (!q.empty()) {
        Node current = q.front();
        q.pop();

        // Print the current node with indentation based on its depth
        cout << setw(current.level * 4) << "" << "Size: " << current.size << ", Level: " << current.level << endl;

        // If the current level is less than the maximum depth, enqueue child nodes
        if (current.level < depth) {
            // Create and enqueue child nodes
            for (int i = 0; i < a; ++i) {
                Node child;
                child.size = current.size / b; // Update problem size
                child.level = current.level + 1; // Increment depth level
```



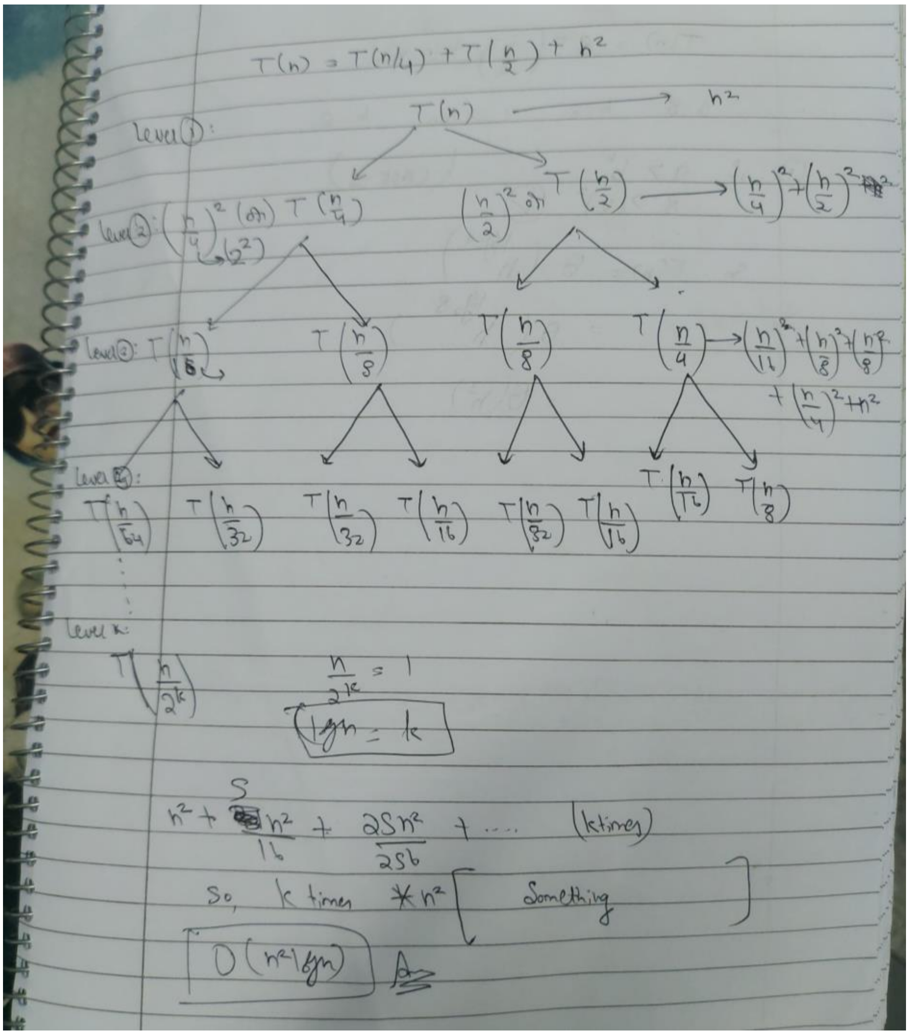
```cpp
                q.push(child);
            }
        }
    }
}

int main() {
    int a, b, depth;

    // Prompt the user for input
    cout << "Enter the number of subproblems (a): ";
    cin >> a;
    cout << "Enter the division factor (b): ";
    cin >> b;
    cout << "Enter the maximum depth of the recursion tree: ";
    cin >> depth;

    // Validate user inputs
    if (a <= 0 || b <= 1 || depth < 0) {
        cout << "Invalid input. Please ensure a > 0, b > 1, and depth >= 0." << endl;
        return 1;
    }

    // Print the recursion tree
    printRecursionTree(a, b, depth);

    return 0;
}
```

Output :



```
cd "/Users/manshu/Documents/DAA ASSIGNMENTS/" && g++ Recursion_Tree.cpp -o Recursion_Tree && "/Users/manshu/Documents/DAA ASSIGNMENTS/"Recursion_Tree
● → Webakriti cd "/Users/manshu/Documents/DAA ASSIGNMENTS/" && g++ Recursion_Tree.cpp -o Recursion_Tree && "/Users/manshu/Documents/DAA ASSIGNMENTS/"Recursi
on_Tree
Recursion_Tree.cpp:41:20: warning: generalized initializer lists are a C++11 extension [-Wc++11-extensions]
            return {0, 0}; // Return a default node
                   ^~~~~~

1 warning generated.
Enter the number of subproblems (a): 2
Enter the division factor (b): 2
Enter the maximum depth of the recursion tree: 3
Size: 100, Level: 0
    Size: 50, Level: 1
    Size: 50, Level: 1
        Size: 25, Level: 2
        Size: 25, Level: 2
        Size: 25, Level: 2
        Size: 25, Level: 2
            Size: 12, Level: 3
            Size: 12, Level: 3
            Size: 12, Level: 3
            Size: 12, Level: 3
            Size: 12, Level: 3
            Size: 12, Level: 3
            Size: 12, Level: 3
            Size: 12, Level: 3
○ → DAA ASSIGNMENTS █
```

Time Complexity :



$T(n) = T(n/4) + T(\frac{n}{2}) + n^2$

$\frac{n}{2^k} = 1$

$\boxed{\lg n = k}$

$n^2 + \frac{5n^2}{16} + \frac{25n^2}{256} + \dots$ (k times)

So, k times $* n^2$ [Something]

$\boxed{O(n^2 \lg n)}$ Ans