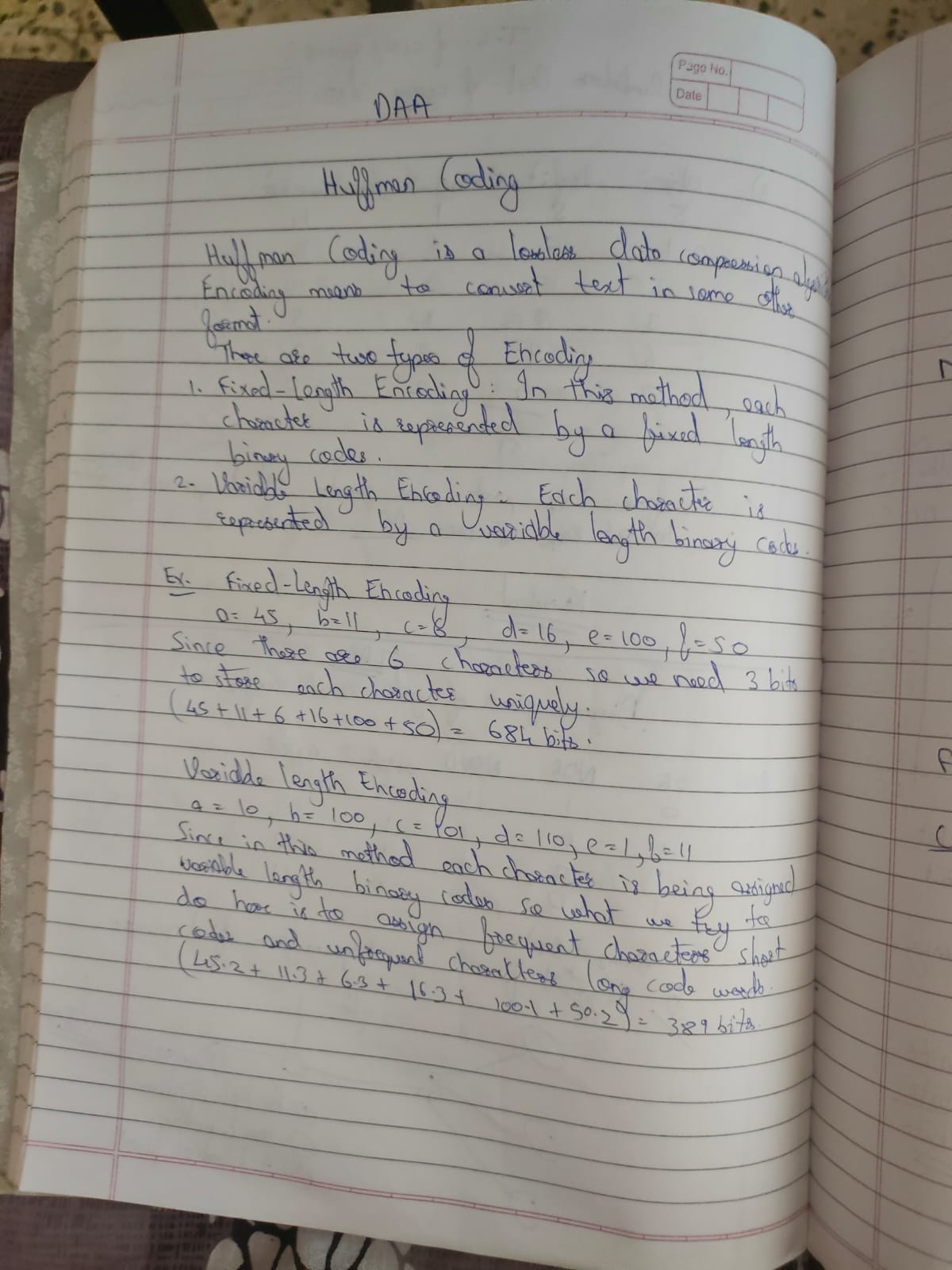
Experiment Number: 6

NAME: Harshwardhan Patil  ROLLNO: 50

CLASS: TY-IT-B BATCH: B1

**Problem Statement:**

**Time complexity analysis of Huffman Encoding using Greedy strategy**



A notebook with writing on it

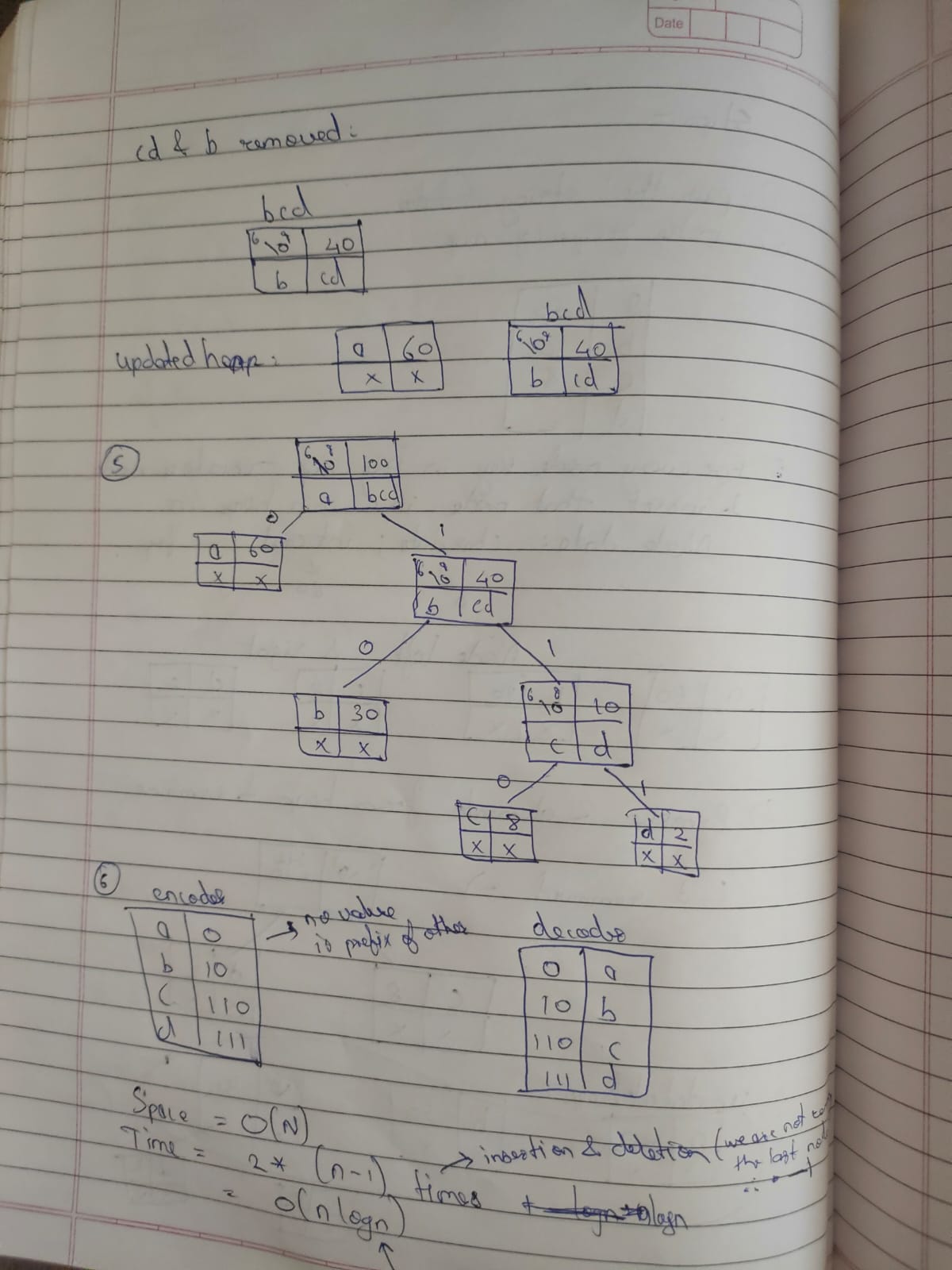
AI-generated content may be incorrect.

A notebook with writing on it

AI-generated content may be incorrect.

A notebook with writing on it

AI-generated content may be incorrect.



A notebook with writing on it

AI-generated content may be incorrect.

import java.util.PriorityQueue;

import java.util.Scanner;

public class HuffmanCoding {

public HuffmanCoding() {

}

public static void generateCodes(HuffmanNode root, String code) {

if (root != null) {

if (root.left == null && root.right == null) {

System.out.println(root.character + " : " + code);

}

generateCodes(root.left, code + "0");

generateCodes(root.right, code + "1");

}

}

public static HuffmanNode buildHuffmanTree(char[] characters, int[] frequencies) {

PriorityQueue<HuffmanNode> pq = new PriorityQueue();

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

pq.add(new HuffmanNode(characters[i], frequencies[i]));

}

while (pq.size() > 1) {

HuffmanNode left = (HuffmanNode) pq.poll();

HuffmanNode right = (HuffmanNode) pq.poll();

HuffmanNode newNode = new HuffmanNode(left.frequency + right.frequency, left, right);

pq.add(newNode);

}

return (HuffmanNode) pq.poll();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of unique characters: ");

int n = scanner.nextInt();

char[] characters = new char[n];

int[] frequencies = new int[n];

System.out.println("Enter characters and their corresponding frequencies:");

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

System.out.print("Character: ");

characters[i] = scanner.next().charAt(0);

System.out.print("Frequency: ");

frequencies[i] = scanner.nextInt();

}

HuffmanNode root = buildHuffmanTree(characters, frequencies);

System.out.println("\nHuffman Codes:");

generateCodes(root, "");

scanner.close();

}

}

class HuffmanNode implements Comparable<HuffmanNode> {

char character;

int frequency;

HuffmanNode left;

HuffmanNode right;

public HuffmanNode(char character, int frequency) {

this.character = character;

this.frequency = frequency;

this.left = this.right = null;

}

public HuffmanNode(int frequency, HuffmanNode left, HuffmanNode right) {

this.character = 0;

this.frequency = frequency;

this.left = left;

this.right = right;

}

public int compareTo(HuffmanNode node) {

return this.frequency - node.frequency;

}

}