

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

Experiment Title: To implement programs on B-Trees and Hash Tables.

Aim/Objective: To understand the concepts and implementation of programs on B-Trees and Hash Tables.

Description: To learn about B-Trees and Hash Tables, operations, and applications. Students will gain experience in implementing these data structures, analyzing their properties, and applying them to solve real-world problems.

Pre-Requisites:

Knowledge: B-Trees, Hash Tables and Collision Resolution Techniques.

Tools: Code Blocks/Eclipse IDE.

Pre-Lab:

1. You are tasked with designing an index for a database using a B-tree of degree t . Given a dataset of keys representing records in a database, implement an algorithm to insert these keys into a B-tree. After constructing the B-tree, perform a series of range queries to fetch records.

Input:

- An integer t representing the degree of the B-tree ($2 \leq t \leq 10$).
- An integer n representing the number of keys to be inserted.
- n space-separated integers representing the keys.
- An integer q representing the number of range queries.
- q pairs of integers $l\ r$ representing the range $[l, r]$.

Output: For each query, output the keys in the range $[l, r]$ in sorted order.

Example

Input:

```
3
10
50 20 30 40 10 60 80 70 90 100
2
30 70
```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	1 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

10 50

Output:

30 40 50 60 70

10 20 30 40 50

• Procedure/Program:

```
import java.util.Scanner;
```

```
class BTreeNode {
    int[] keys;
    int t;
    BTreeNode[] C;
    int n;
    boolean leaf;

    BTreeNode(int t, boolean leaf) {
        this.t = t;
        this.leaf = leaf;
        this.keys = new int[2 * t - 1];
        this.C = new BTreeNode[2 * t];
        this.n = 0;
    }
}
```

```
public class Main {
    static BTreeNode createNode(int t, boolean leaf) {
        return new BTreeNode(t, leaf);
    }

    static void splitChild(BTreeNode parent, int i, BTreeNode child) {
        BTreeNode newChild = createNode(child.t, child.leaf);
        newChild.n = child.t - 1;
        for (int j = 0; j < child.t - 1; j++)
            newChild.keys[j] = child.keys[j + child.t];
        if (!child.leaf) {
            for (int j = 0; j < child.t; j++)
                newChild.C[j] = child.C[j + child.t];
        }
    }
}
```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	2 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

```

    }
    child.n = child.t - 1;
    for (int j = parent.n; j >= i + 1; j--)
        parent.C[j + 1] = parent.C[j];
    parent.C[i + 1] = newChild;
    for (int j = parent.n - 1; j >= i; j--)
        parent.keys[j + 1] = parent.keys[j];
    parent.keys[i] = child.keys[child.t - 1];
    parent.n++;
}

static void insertNonFull(BTreeNode node, int key) {
    int i = node.n - 1;
    if (node.leaf) {
        while (i >= 0 && key < node.keys[i]) {
            node.keys[i + 1] = node.keys[i];
            i--;
        }
        node.keys[i + 1] = key;
        node.n++;
    } else {
        while (i >= 0 && key < node.keys[i])
            i--;
        i++;
        if (node.C[i].n == 2 * node.t - 1) {
            splitChild(node, i, node.C[i]);
            if (key > node.keys[i])
                i++;
        }
        insertNonFull(node.C[i], key);
    }
}

static void insert(BTreeNode[] root, int key) {
    if (root[0].n == 2 * root[0].t - 1) {
        BTreeNode newRoot = createNode(root[0].t, false);
        newRoot.C[0] = root[0];
        splitChild(newRoot, 0, root[0]);
    }
}

```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	3 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

```

        int i = 0;
        if (newRoot.keys[0] < key)
            i++;
        insertNonFull(newRoot.C[i], key);
        root[0] = newRoot;
    } else {
        insertNonFull(root[0], key);
    }
}

static void rangeQuery(BTreeNode node, int l, int r) {
    int i;
    for (i = 0; i < node.n; i++) {
        if (node.keys[i] >= l && node.keys[i] <= r)
            System.out.print(node.keys[i] + " ");
    }
    if (!node.leaf) {
        for (int j = 0; j <= node.n; j++) {
            if (j < node.n && node.keys[j] >= l)
                rangeQuery(node.C[j], l, r);
            else if (j == node.n)
                rangeQuery(node.C[j], l, r);
        }
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int t = scanner.nextInt();
    int n = scanner.nextInt();
    BTreeNode[] root = new BTreeNode[1];
    root[0] = createNode(t, true);
    for (int i = 0; i < n; i++) {
        int key = scanner.nextInt();
        insert(root, key);
    }
    int q = scanner.nextInt();
    for (int i = 0; i < q; i++) {

```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	4 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

```

        int l = scanner.nextInt();
        int r = scanner.nextInt();
        rangeQuery(root[0], l, r);
        System.out.println();
    }
    scanner.close();
}
}

```

- **Data and Results:**

Data: The dataset consists of keys to be inserted into B-tree.

Result: The B-tree is successfully constructed and range queries executed.

- **Analysis and Inferences:**

Analysis: Efficient insertion and range queries using the B-tree structure.

Inferences: B-tree improves performance for insertions and range queries in databases.

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	5 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

2. Given a string S of length N consisting of only lower-case English alphabets, you will be asked to process Q queries over it . In each query you will be given two lower case characters X and Y . Your task is to find out the number of such substrings of the the string S which have the characters X and Y on either of its end points, both $X...Y$ and $Y...X$ are considered to be valid.

Note : Substrings length should be greater than 1.

Input Format

- The first line of the input will contain N , the length of the string.
- Next line will contain as string of length N . Next line will will contain Q , the number of queries.
- Then Q subsequent lines will contain two lowercase characters X and Y separated by a space.

Constraints

$$1 \leq N \leq 10^6$$

$$1 \leq Q \leq 10^3$$

Output Format

For each query, output the answer in a separate line.

Sample Input 0

```
5
aacbb
2
a c
a b
```

Sample Output 0

```
2
4
```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	6 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

• **Procedure/Program:**

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int N = scanner.nextInt();
        String S = scanner.next();
        int Q = scanner.nextInt();

        for (int i = 0; i < Q; i++) {
            char X = scanner.next().charAt(0);
            char Y = scanner.next().charAt(0);
            int count = 0;

            for (int j = 0; j < N; j++) {
                if (S.charAt(j) == X) {
                    for (int k = j + 1; k < N; k++) {
                        if (S.charAt(k) == Y) {
                            count++;
                        }
                    }
                } else if (S.charAt(j) == Y) {
                    for (int k = j + 1; k < N; k++) {
                        if (S.charAt(k) == X) {
                            count++;
                        }
                    }
                }
            }
            System.out.println(count);
        }
        scanner.close();
    }
}
```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	7 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

- **Data and Results:**

- **Data:** The problem involves counting substrings with specific start and end characters.
- **Result:** The output shows the number of valid substrings for each query.

- **Analysis and Inferences:**

- **Analysis:** The solution uses brute-force searching, iterating through possible pairs.
- **Inferences:** Optimization is needed for large strings to handle maximum constraints effectively.

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

In-Lab:

1. You are managing a large database system where quick insertion and search operations are essential to handle high-volume transactions. The system uses a **B-Tree** to index its data for efficient disk-based storage and retrieval.

Operations in Context:

1. **Insert Operation:**

A new record with key kkk (e.g., Employee ID) needs to be added to the database.

Example: Adding a new Employee ID 12345.

2. **Search Operation:**

A user queries the system to check if a record with a specific Employee ID exists.

Example: Searching for Employee ID 67890.

Sample Input :

2 6

INSERT 10123

INSERT 54321

INSERT 67890

SEARCH 10123

SEARCH 99999

SEARCH 67890

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	9 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

Sample Output:

YES

NO

YES

• Procedure/Program:

```
import java.util.Scanner;
```

```
class BTreeNode {
    int[] keys;
    int t;
    BTreeNode[] C;
    int n;
    boolean leaf;

    BTreeNode(int t, boolean leaf) {
        this.t = t;
        this.leaf = leaf;
        this.keys = new int[2 * t - 1];
        this.C = new BTreeNode[2 * t];
        this.n = 0;
    }
}
```

```
public class Main {
    static BTreeNode createNode(int t, boolean leaf) {
        return new BTreeNode(t, leaf);
    }

    static void insertNonFull(BTreeNode node, int k) {
        int i = node.n - 1;
        if (node.leaf) {
            while (i >= 0 && node.keys[i] > k) {
                node.keys[i + 1] = node.keys[i];
                i--;
            }
        }
    }
}
```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	10 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

```

    }
    node.keys[i + 1] = k;
    node.n++;
} else {
    while (i >= 0 && node.keys[i] > k) i--;
    if (node.C[i + 1].n == 2 * node.t - 1) {
        BTreeNode temp = createNode(node.t, node.C[i + 1].leaf);
        BTreeNode child = node.C[i + 1];
        temp.n = node.t - 1;
        for (int j = 0; j < node.t - 1; j++) temp.keys[j] = child.keys[j + node.t];
        if (!child.leaf) {
            for (int j = 0; j < node.t; j++) temp.C[j] = child.C[j + node.t];
        }
        child.n = node.t - 1;
        for (int j = node.n; j >= i + 1; j--) node.C[j + 1] = node.C[j];
        node.C[i + 1] = temp;
        for (int j = node.n - 1; j >= i; j--) node.keys[j + 1] = node.keys[j];
        node.keys[i] = child.keys[node.t - 1];
        node.n++;
        if (node.keys[i] < k) i++;
    }
    insertNonFull(node.C[i + 1], k);
}
}

static void insert(BTreeNode[] root, int k) {
    if (root[0].n == 2 * root[0].t - 1) {
        BTreeNode newNode = createNode(root[0].t, false);
        newNode.C[0] = root[0];
        newNode.n = 0;
        root[0] = newNode;
        insertNonFull(newNode, k);
    } else {
        insertNonFull(root[0], k);
    }
}

static boolean search(BTreeNode root, int k) {

```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	11 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

```

int i = 0;
while (i < root.n && k > root.keys[i]) i++;
if (i < root.n && root.keys[i] == k) return true;
if (root.leaf) return false;
return search(root.C[i], k);
}

```

```

public static void main(String[] args) {
    BTreeNode[] root = new BTreeNode[1];
    root[0] = createNode(3, true);
    Scanner scanner = new Scanner(System.in);
    String command;
    int id;

    while (scanner.hasNext()) {
        command = scanner.next();
        id = scanner.nextInt();
        if (command.equals("INSERT")) {
            insert(root, id);
        } else if (command.equals("SEARCH")) {
            System.out.print(search(root[0], id) ? "YES " : "NO ");
        }
    }
    scanner.close();
}
}

```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	12 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

- **Data and Results:**

Data: Inserting and searching for Employee IDs in a database system.

Result: Efficient insertion and search operations with B-Tree indexing.

- **Analysis and Inferences:**

Analysis: B-Tree ensures quick data insertion and fast search queries.

Inferences: B-Tree is optimal for large datasets and high-volume transactions.

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

2. Harold is a kidnapper who wrote a ransom note, but now he is worried it will be traced back to him through his handwriting. He found a magazine and wants to know if he can cut out whole words from it and use them to create an untraceable replica of his ransom note. The words in his note are case-sensitive and he must use only whole words available in the magazine. He cannot use substrings or concatenation to create the words he needs. Given the words in the magazine and the words in the ransom note, print Yes if he can replicate his ransom note *exactly* using whole words from the magazine; otherwise, print No.

Example

magazine = "attack at dawn" note = "Attack at dawn"

The magazine has all the right words, but there is a case mismatch. The answer is No .

Function Description

Complete the checkMagazine function in the editor below. It must print Yes if the note can be formed using the magazine, or No.

checkMagazine has the following parameters:

- string magazine[m]: the words in the magazine
- string note[n]: the words in the ransom note

Prints

- *string*: either Yes or No. no return value is expected.

Input Format

The first line contains two space-separated integers m and n, the numbers of words in the magazine and the note, respectively.

The second line contains m space-separated strings, each magazine[i].

The third line contains n space-separated strings, each note[i].

Constraints

- $1 \leq m \leq 30000$
- $1 \leq \text{lengthsof magazine[i] and note[i]} \leq 5$.
- Each word consists of English alphabetic letters (i.e., a to z , A to Z).

Sample Input 0

6 4

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	14 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

give me one grand today night

give one grand today

Sample Output 0

Yes

Sample Input 1

6 5

two times three is not four

two times two is four

Sample Output 1

No

• Procedure/Program:

```
import java.util.Arrays;
import java.util.Scanner;

public class Main {
    private static final int MAX_WORD_LENGTH = 5;

    public static void checkMagazine(String[] magazine, String[] note) {
        Arrays.sort(magazine);
        Arrays.sort(note);

        int i = 0, j = 0;

        while (i < magazine.length && j < note.length) {
            if (magazine[i].equals(note[j])) {
                i++;
                j++;
            } else if (magazine[i].compareTo(note[j]) < 0) {
                i++;
            } else {
                System.out.println("No");
                return;
            }
        }
    }
}
```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	15 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

```

    }

    if (j == note.length) {
        System.out.println("Yes");
    } else {
        System.out.println("No");
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int m = scanner.nextInt();
    int n = scanner.nextInt();
    scanner.nextLine(); // Consume the newline

    String[] magazine = new String[m];
    String[] note = new String[n];

    for (int i = 0; i < m; i++) {
        magazine[i] = scanner.nextLine();
    }

    for (int i = 0; i < n; i++) {
        note[i] = scanner.nextLine();
    }

    checkMagazine(magazine, note);
    scanner.close();
}
}

```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	16 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

- **Data and Results:**

Data

The data consists of words from the magazine and ransom note.

Result

The result determines if the ransom note can be formed.

- **Analysis and Inferences:**

Analysis

Analysis includes comparing word counts and matching them between lists.

Inferences

Inferences show if the magazine contains enough words for note.

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	17 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

Post-Lab:

1. A library uses a B-Tree to manage its catalogue of books. Each book is identified by a unique ISBN number.

Operations:

Insert: Add a new book to the catalog.

Search: Check if a book is available using its ISBN.

• Procedure/Program:

```
import java.util.Arrays;
```

```
class BTreeNode {
```

```
    int n;
```

```
    String[] keys;
```

```
    BTreeNode[] children;
```

```
    boolean leaf;
```

```
    static final int MAX = 3;
```

```
    BTreeNode(boolean leaf) {
```

```
        this.leaf = leaf;
```

```
        this.n = 0;
```

```
        this.keys = new String[MAX];
```

```
        this.children = new BTreeNode[MAX + 1];
```

```
    }
```

```
}
```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	18 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

```

public class Main {
    static BTreeNode createNode(boolean leaf) {
        return new BTreeNode(leaf);
    }

    static void insertNonFull(BTreeNode node, String isbn) {
        int i = node.n - 1;
        if (node.leaf) {
            while (i >= 0 && isbn.compareTo(node.keys[i]) < 0) {
                node.keys[i + 1] = node.keys[i];
                i--;
            }
            node.keys[i + 1] = isbn;
            node.n++;
        } else {
            while (i >= 0 && isbn.compareTo(node.keys[i]) < 0) {
                i--;
            }
            i++;
            if (node.children[i].n == BTreeNode.MAX) {
                BTreeNode newChild = createNode(node.children[i].leaf);
                BTreeNode oldChild = node.children[i];
                newChild.n = BTreeNode.MAX / 2;
                node.n++;
                for (int j = node.n - 1; j > i; j--) {
                    node.children[j + 1] = node.children[j];
                }
                node.children[i + 1] = newChild;
            }
        }
    }
}

```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	19 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

```

        for (int j = 0; j < newChild.n; j++) {
            newChild.keys[j] = oldChild.keys[j + BTreeNode.MAX / 2];
        }
        oldChild.n = BTreeNode.MAX / 2;
        insertNonFull(node.children[i], isbn);
    } else {
        insertNonFull(node.children[i], isbn);
    }
}
}

```

```

static void insert(BTreeNode[] root, String isbn) {
    if (root[0].n == BTreeNode.MAX) {
        BTreeNode newRoot = createNode(false);
        newRoot.children[0] = root[0];
        insertNonFull(newRoot, isbn);
        root[0] = newRoot;
    } else {
        insertNonFull(root[0], isbn);
    }
}

```

```

static boolean search(BTreeNode node, String isbn) {
    int i = 0;
    while (i < node.n && isbn.compareTo(node.keys[i]) > 0) {
        i++;
    }
    if (i < node.n && isbn.equals(node.keys[i])) {

```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	20 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

```

        return true;
    }
    if (node.leaf) {
        return false;
    }
    return search(node.children[i], isbn);
}

```

```

public static void main(String[] args) {
    BTreeNode[] root = new BTreeNode[1];
    root[0] = createNode(true);
    insert(root, "978-3-16-148410-0");
    insert(root, "978-1-60309-452-8");
    insert(root, "978-0-262-13472-9");

```

```

        System.out.println("Searching for ISBN 978-3-16-148410-0: " + (search(root[0],
"978-3-16-148410-0") ? "Found" : "Not Found"));

```

```

        System.out.println("Searching for ISBN 978-1-23456-789-0: " + (search(root[0],
"978-1-23456-789-0") ? "Found" : "Not Found"));

```

```

    }
}

```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	21 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

- **Data and Results:**

Data

The dataset includes various ISBN numbers for books in the catalog.

Result

The search operation successfully finds or misses books by ISBN.

- **Analysis and Inferences:**

Analysis

The B-Tree structure optimizes search time and insertion efficiency.

Inferences

The B-Tree balances nodes for fast lookups and efficient insertion.

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	22 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

2. A university system uses a hash table to manage student IDs and their data.

Operations:

Insert: Add a new student to the database.

Search: Retrieve a student's information using their student ID.

• **Procedure/Program:**

```
import java.util.LinkedList;
```

```
class Student {
    int id;
    String name;

    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

```
class HashTable {
    private static final int TABLE_SIZE = 100;
    private LinkedList<Student>[] table;

    @SuppressWarnings("unchecked")
    public HashTable() {
        table = new LinkedList[TABLE_SIZE];
        for (int i = 0; i < TABLE_SIZE; i++) {
            table[i] = new LinkedList<>();
        }
    }

    private int hash(int id) {
        return id % TABLE_SIZE;
    }

    public void insert(int id, String name) {
        int index = hash(id);
        table[index].add(new Student(id, name));
    }
}
```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	23 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

```

public Student search(int id) {
    int index = hash(id);
    for (Student student : table[index]) {
        if (student.id == id) {
            return student;
        }
    }
    return null;
}

public class Main {
    public static void main(String[] args) {
        HashTable hashTable = new HashTable();
        hashTable.insert(1, "Alice");
        hashTable.insert(2, "Bob");

        Student student = hashTable.search(1);
        if (student != null) {
            System.out.println("Found: " + student.name);
        } else {
            System.out.println("Student not found.");
        }
    }
}

```

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	24 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

- **Data and Results:**

Data

The system uses a hash table to store student data efficiently, where each student is identified by a unique ID.

Result

The program allows the insertion of students and retrieval based on their IDs.

- **Analysis and Inferences:**

Analysis

Hashing ensures quick lookup and insertion with minimal collisions.

Inferences

The hash table effectively supports scalable student data management and retrieval.

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	25 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

• **Sample VIVA-VOCE Questions (In-Lab):**

1. What are the time complexities of search, insertion, and deletion in a B-Tree?

- **Search:** $O(\log_t N)$
- **Insertion:** $O(\log_t N)$
- **Deletion:** $O(\log_t N)$

2. Compare and contrast B-Trees with other balanced trees like AVL or Red-Black Trees.

- **B-Trees:** Efficient for large datasets, used in databases/filesystems.
- **AVL:** Strict balancing, faster lookups, but costly insertions/deletions.
- **Red-Black:** More relaxed balancing, faster insertions/deletions, slightly slower lookups.

3. How does the degree t affect the structure and performance of a B-Tree?

- A higher t leads to fewer tree levels, better performance, and fewer disk accesses.

4. Differentiate between open addressing & separate chaining for collision resolution.

- **Open addressing:** Stores elements directly in the hash table, requires probing.
- **Separate chaining:** Uses linked lists for collisions, more space but fewer clustering issues.

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	26 Page

Experiment #4		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

5. How does a hash table differ from a binary search tree in terms of use cases?

- Hash table: $O(1)$ average time, great for fast lookups.
- BST: $O(\log N)$ for balanced trees, ideal for sorted data and range queries.

Evaluator Remark (if Any):	Marks Secured ____ out of 50
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Course Title	Advanced Algorithms & Data Structures	ACADEMIC YEAR: 2024-25
Course Code	23CS03HF	27 Page