# DSA Practicals - Atharva Jadhav

## Practical 1: Telephone Book Hashing (Python)

```
class HashTable:
    def __init__(self, size): self.table = [None]*size
    def hash(self, key): return key % len(self.table)

    def insert_linear(self, key):
        i = self.hash(key)
        while self.table[i] is not None:
            i = (i + 1) % len(self.table)
        self.table[i] = key

    def show(self):
        print(self.table)

h = HashTable(10)
for k in [23, 43, 13, 27]: h.insert_linear(k)
h.show()
```

## Practical 4: Set ADT (Python)

```
class MySet:
    def __init__(self): self.data = []

    def add(self, val):
        if val not in self.data: self.data.append(val)

    def remove(self, val):
        if val in self.data: self.data.remove(val)

    def contains(self, val): return val in self.data
    def size(self): return len(self.data)
    def show(self): print(self.data)

s = MySet()
s.add(1); s.add(2); s.add(3); s.remove(2)
s.show()
```

## Practical 5: Book Tree (C++)

```
#include <iostream>
using namespace std;

struct Node {
    string name;
    Node* left;
    Node* right;
};

Node* createNode(string name) {
    Node* n = new Node();
    n->name = name;
    n->left = n->right = nullptr;
    return n;
}

void printTree(Node* root) {
    if (!root) return;
    cout << root->name << endl;
    printTree(root->left);
    printTree(root->right);
}

int main() {
    Node* book = createNode("Book");
    book->left = createNode("Chapter 1");
    book->right = createNode("Chapter 2");
    book->left->left = createNode("Section 1.1");
    book->left->right = createNode("Section 1.2");
    printTree(book);
    return 0;
}
```

## Practical 6: BST Insert and Inorder (C++)

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* left, *right;
    Node(int d) : data(d), left(NULL), right(NULL) {}
};

Node* insert(Node* root, int val) {
    if (!root) return new Node(val);
    if (val < root->data) root->left = insert(root->left, val);
    else root->right = insert(root->right, val);
    return root;
}

void inorder(Node* root) {
    if (!root) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

int main() {
    Node* root = NULL;
    int arr[] = { 10, 5, 15, 3 };
    for (int v : arr) root = insert(root, v);
    inorder(root);
    return 0;
}
```

## Practical 9: Threaded Binary Tree - Basic (C++)

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node *left, *right;
    bool lthread, rthread;
};

Node* create(int data) {
    Node* n = new Node();
    n->data = data;
    n->left = n->right = NULL;
    n->lthread = n->rthread = true;
    return n;
}

int main() {
    Node* root = create(10);
    root->left = create(5);
    root->right = create(15);
    cout << "Root: " << root->data;
    return 0;
}
```

## Practical 13: Graph BFS (C++)

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

void bfs(int start, vector<vector<int>>& adj, vector<bool>& visited) {
    queue<int> q;
    visited[start] = true;
    q.push(start);

    while (!q.empty()) {
        int v = q.front(); q.pop();
        cout << v << " ";
        for (int u : adj[v])
            if (!visited[u]) {
                visited[u] = true;
                q.push(u);
            }
    }
}
```

```
int main() {
    vector<vector<int>> adj = {{}, {2,3}, {1,4}, {1}, {2}};
    vector<bool> visited(5, false);
    bfs(1, adj, visited);
    return 0;
}
```

## Practical 14: Flight Path Graph (C++)

```
#include <iostream>
using namespace std;

int main() {
    int graph[4][4] = {
        {0,1,1,0},
        {1,0,1,0},
        {1,1,0,1},
        {0,0,1,0}
    };
    cout << "Adjacency Matrix:\n";
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++)
            cout << graph[i][j] << " ";
        cout << endl;
    }
    return 0;
}
```

## Practical 18: Optimal BST Cost Estimate (C++)

```
#include <iostream>
using namespace std;

int main() {
    int keys[] = {10, 20, 30};
    float prob[] = {0.2, 0.5, 0.3};
    float cost = 0;
    for (int i = 0; i < 3; i++)
        cost += (i+1) * prob[i];
    cout << "Approx. search cost: " << cost << endl;
    return 0;
}
```

## Practical 19: Simple Dictionary (C++)

```
#include <iostream>
#include <map>
using namespace std;

int main() {
    map<string, string> dict;
    dict["apple"] = "A fruit";
    dict["book"] = "A source of knowledge";

    for (auto& x : dict)
        cout << x.first << " : " << x.second << endl;

    cout << "Search 'book': " << dict["book"] << endl;
    return 0;
}
```

## Practical 22: Heap Max/Min (C++)

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    vector<int> marks = {70, 85, 60, 90};
    make_heap(marks.begin(), marks.end());
    cout << "Max marks: " << marks.front() << endl;

    sort_heap(marks.begin(), marks.end());
    cout << "Min marks: " << marks.front() << endl;
    return 0;
}
```

## Practical 23: Student File Seq Access (C++)

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream f("student.txt");
    f << "101 Atharva\n102 Raj\n"; f.close();

    ifstream fin("student.txt");
    string line;
    while (getline(fin, line)) cout << line << endl;
    return 0;
}
```

## Practical 24: Employee File Indexed (C++)

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream f("emp.txt");
    f << "201 John\n202 Aman\n"; f.close();
    ifstream fin("emp.txt");
    string line;
    while (getline(fin, line)) cout << line << endl;
    return 0;
}
```