Date : 17.03.2025
Lakshana Baskaran
24011103026

LAB - 8
Tree Data Structure

QUESTION 1:

A. There are n block towers, numbered from 1 to n. The i-th tower consists of a_i blocks.

In one move, you can move one block from tower i to tower j, but only if $a_i > a$▣. That move increases a▣ by 1 and decreases a_i by 1.
You can perform as many moves as you like (possibly zero).

What's the largest amount of blocks you can have on tower 1 after the moves?

Input:

- The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.
- The first line of each test case contains a single integer n ($2 \le n \le 2 \times 10^5$) — the number of towers.
- The second line contains n integers $a_1, a_2, ..., a$▣ ($1 \le a_i \le 10^9$) — the number of blocks on each tower.
- The sum of n over all test cases doesn't exceed 2×10^5.

SOURCE CODE:

```
//what is the maximum number of blocks possible in tower 1?
#include <iostream>
using namespace std;

void tower_one_blocks(int*, int);


int main() {
    int t;
    cin >> t;
    int* tes = (int*) malloc (t*sizeof(int));
    tower_one_blocks(tes, t);

    for (int j = 0; j < t; ++j) {
```

Date : 17.03.2025
Lakshana Baskaran
24011103026

```
            cout << "Test Case " << j+1 << ": " << tes[j] << "\n";
        }
        cout << endl;

}

//calculates the maximum number of blocks that can be possibly stored
in the first tower
//given the condition that a block can only be moved for a tower with
higher number of blockas to a tower having lower number of blocks
//for all test cases
void tower_one_blocks(int* tesarr, int t) {
    int n, blk;
    int arr[n];
    for (int i = 0; i < t; ++i) {
        cin >> n;
        for (int j = 0; j < n; ++j) {
            cin >> arr[j];
        }

        blk = arr[0];
        for (int j = 1; j < n; ++j) {
            while (true) {
                if (blk >= arr[j]) {
                    break;
                }
                arr[0]++;
                arr[j]--;
                blk = arr[0];
            }
        }
        tesarr[i] = blk;
    }
}
```

Date : 17.03.2025
Lakshana Baskaran
24011103026

OUTPUT:

```
lemon@jupiter:~/workspace/college/DSA/Lab-7$ g++ -o out towers.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-7$ ./out
3
2
1 1000000
3
1 2 3
4
4 3 2 1
Test Case 1: 500001
Test Case 2: 3
Test Case 3: 4
```

Date : 17.03.2025
Lakshana Baskaran
24011103026

QUESTION 2:

Write a C++ menu-driven program to implement the Binary Tree ADT using a linked list. Maintain proper boundary conditions and follow good coding practices. The Binary Tree ADT must support the following operations:

1. Insert a node
2. Delete a node
3. Preorder traversal
4. Inorder traversal
5. Postorder traversal
6. Exit

SOURCE CODE:

```
//Implementation of tree ADT using linked lists
#include <cstddef>
#include <cstdio>
#include <stdio.h>
#include <stdbool.h>
#include <iostream>
using namespace std;

class tree{
    private:
        struct TreeNode {
            int data;
            TreeNode *left;
            TreeNode *right;
        };
        TreeNode *root;

    public:
        TreeNode* create_node(int);
        void Inorder_traversal();
        void inorder_traversal(TreeNode*);

        void Postorder_traversal();
        void postorder_traversal(TreeNode*);

        void Preorder_traversal();
        void preorder_traversal(TreeNode*);
```

```
        void insert(int);
        bool search(int);

        tree() {
            root = NULL;
        }

};

//Inorder traversal of tree
void tree::Inorder_traversal() {
    inorder_traversal(root);
}

void tree::inorder_traversal(TreeNode* root) {
    if (root == NULL) {
        return;
    }
    inorder_traversal(root->left);
    cout << root->data << "  ";
    inorder_traversal(root->right);
}

//Postorder traversal of tree
void tree::Postorder_traversal() {
    postorder_traversal(root);
}

void tree::postorder_traversal(TreeNode* root) {
    if (root == NULL) {
        return;
    }
    postorder_traversal(root->left);
    postorder_traversal(root->right);
    cout << root->data << "  ";
}

//Preorder traversal of tree
void tree::Preorder_traversal() {
    preorder_traversal(root);
}

void tree::preorder_traversal(TreeNode* root) {
```

```
    if (root == NULL) {
        return;
    }
    cout << root->data << "  ";
    preorder_traversal(root->left);
    preorder_traversal(root->right);
}


//level order insertion
void tree::insert(int x) {
    TreeNode* newNode = new TreeNode;
    newNode->data = x;
    newNode->left = nullptr;
    newNode->right = nullptr;

    if (!root) {
        root = newNode;
        return;
    }

    TreeNode* temp = root;
    while (true) {
        if (!temp->left) {
            temp->left = newNode;
            return;
        } else if (!temp->right) {
            temp->right = newNode;
            return;
        } else {
            temp = temp->left;
        }
    }
}

//search for a node in tree
bool tree::search(int key) {
    TreeNode* temp = root;
    while (temp) {
        if (temp->data == key)
            return true;
        if (temp->left)
            temp = temp->left;
```

```
        else if (temp->right)
            temp = temp->right;
        else
            break;
    }

    if (temp->data == key) {
        return true;
    }
    return false;
}

int main() {
    tree T;

    int x, choice = 0;

    printf("MENU\n1 - Insert\n2 - Inorder traversal\n3 - Postorder
traversal\n4 - Preorder traversal\n5 - Search\n6 - Exit\n");
    while (true) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter node to be inserted: ");
                scanf("%d", &x);
                T.insert(x);
                break;

            case 2:
                cout << "Inorder traversal: ";
                T.Inorder_traversal();
                printf("\n");
                break;

            case 3:
                cout << "Postorder traversal: ";
                T.Postorder_traversal();
                printf("\n");
                break;

            case 4:
                cout << "Preorder traversal: ";
```

```
                T.Preorder_traversal();
                printf("\n");
                break;

            case 5:
                printf("Enter node to search for: ");
                scanf("%d", &x);
                if (T.search(x)) {
                    printf("Node found\n");
                }
                else {
                    printf("Element is NOT found\n");
                }
                printf("\n");
                break;

            case 6:
                printf("Exiting...\n");
                return 0;
                break;

            default:
                printf("\nInvalid choice. Enter again.\n");
                break;
        }
    }
    return 0;
}
```

Date : 17.03.2025
Lakshana Baskaran
24011103026

SOURCE CODE:

```
lemon@jupiter:~/workspace/college/DSA/Lab-7$ g++ -o out tree.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-7$ ./out
MENU
1 - Insert
2 - Inorder traversal
3 - Postorder traversal
4 - Preorder traversal
5 - Search
6 - Exit

Enter your choice: 1
Enter node to be inserted: 2

Enter your choice: 1
Enter node to be inserted: 5

Enter your choice: 1
Enter node to be inserted: 4

Enter your choice: 1
Enter node to be inserted: 8

Enter your choice: 2
Inorder traversal: 8  5  2  4

Enter your choice: 3
Postorder traversal: 8  5  4  2

Enter your choice: 4
Preorder traversal: 2  5  8  4

Enter your choice: 5
Enter node to search for: 5
Node found


Enter your choice: 5
Enter node to search for: 9
Element is NOT found


Enter your choice: 9

Invalid choice. Enter again.

Enter your choice: 6
Exiting...
lemon@jupiter:~/workspace/college/DSA/Lab-7$
```

—