1. Programs for summation of series 1+X+X^2+X^3+… with different time complexities

**(a) Using Loop – Time Complexity: O(n)**

#include <stdio.h>

#include <math.h>

int main () {

int x, n;

double sum = 0;

printf("Enter the value of x and n: ");

scanf("%d %d", &x, &n);

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

sum += pow (x, i);

}

Printf ("Sum of the series (loop) = %.2lf\n", sum);

return 0;

}

**b) Using Formula – Time Complexity: O(1)**

Geometric Series Formula:

S = x^n+1 -1/x-1

If x ! =1

#include <stdio.h>

#include <math.h>

int main () {

int x, n;

double sum;

printf("Enter the value of x and n: ");

scanf("%d %d", &x, &n);

if (x == 1)

sum = n + 1;

else

sum = (pow (x, n + 1) - 1) / (x - 1);

printf("Sum of the series (formula) = %.2lf\n", sum);

return 0;

}

**(c) Using Recursion – Time Complexity: O(n)**

#include <stdio.h>

#include <math.h>

double recursive Sum(int x, int n) {

if(n == 0)

return 1;

return pow(x, n) + recursive Sum(x, n - 1);

}

int main () {

int x, n;

printf("Enter the value of x and n: ");

scanf("%d %d", &x, &n);

double sum = recursive Sum(x, n);

printf("Sum of the series (recursion) = %.2lf\n", sum);

return 0;

}

1. Create a Binary Search Tree and perform the insertion, deletion operations.

#include <stdio.h>

#include <stdlib.h>

// Structure of a node

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Create a new node

struct Node\* create Node(int value) {

struct Node\* new Node = (struct Node\*) malloc(sizeof(struct Node));

new Node->data = value;

new Node->left = new Node->right = NULL;

return new Node;

}

// Insertion

struct Node\* insert (struct Node\* root, int value) {

if (root == NULL)

return create Node(value);

if (value < root->data)

root->left = insert(root->left, value);

else if (value > root->data)

root->right = insert(root->right, value);

return root;

}

// Find minimum value node

struct Node\* findMin(struct Node\* node) {

while(node->left!= NULL)

node = node->left;

return node;

}

// Deletion

struct Node\* delete Node(struct Node\* root, int key) {

if (root == NULL)

return root;

if (key < root->data)

root->left = delete Node(root->left, key);

else if (key > root->data)

root->right = delete Node(root->right, key);

else {

if(root->left == NULL) {

struct Node\* temp = root->right;

free(root);

return temp;

} else if(root->right == NULL) {

struct Node\* temp = root->left;

free(root);

return temp;

}

struct Node\* temp = findMin(root->right);

root->data = temp->data;

root->right = delete Node(root->right, temp->data);

}

return root;

}

// In-order traversal

void inorder(struct Node\* root) {

if (root != NULL) {

inorder(root->left);

printf("%d ", root->data);

inorder(root->right);

}

}

int main () {

struct Node\* root = NULL;

root = insert (root, 50);

insert (root, 30);

insert (root, 70);

insert (root, 20);

insert (root, 40);

insert (root, 60);

insert (root, 80);

printf("Inorder traversal of BST: ");

inorder(root);

printf("\nDeleting 20\n");

root = delete Node(root, 20);

printf("Inorder after deletion: ");

inorder(root);

printf("\nDeleting 30\n");

root = delete Node(root, 30);

printf("Inorder after deletion: ");

inorder(root);

printf("\nDeleting 50\n");

root = delete Node(root, 50);

printf("Inorder after deletion: ");

inorder(root);

return 0;

}