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

complexities.

Method 1:

#include<stdio.h>

#include<math.h>

int main(){

int a,n,r;

printf("Enter the 1st element: \n");

scanf("%d", &a);

printf("Enter the common ratio value: \n");

scanf("%d", &r);

printf("Enter the no. of terms in the series: \n");

scanf("%d", &n);

int sum=0;

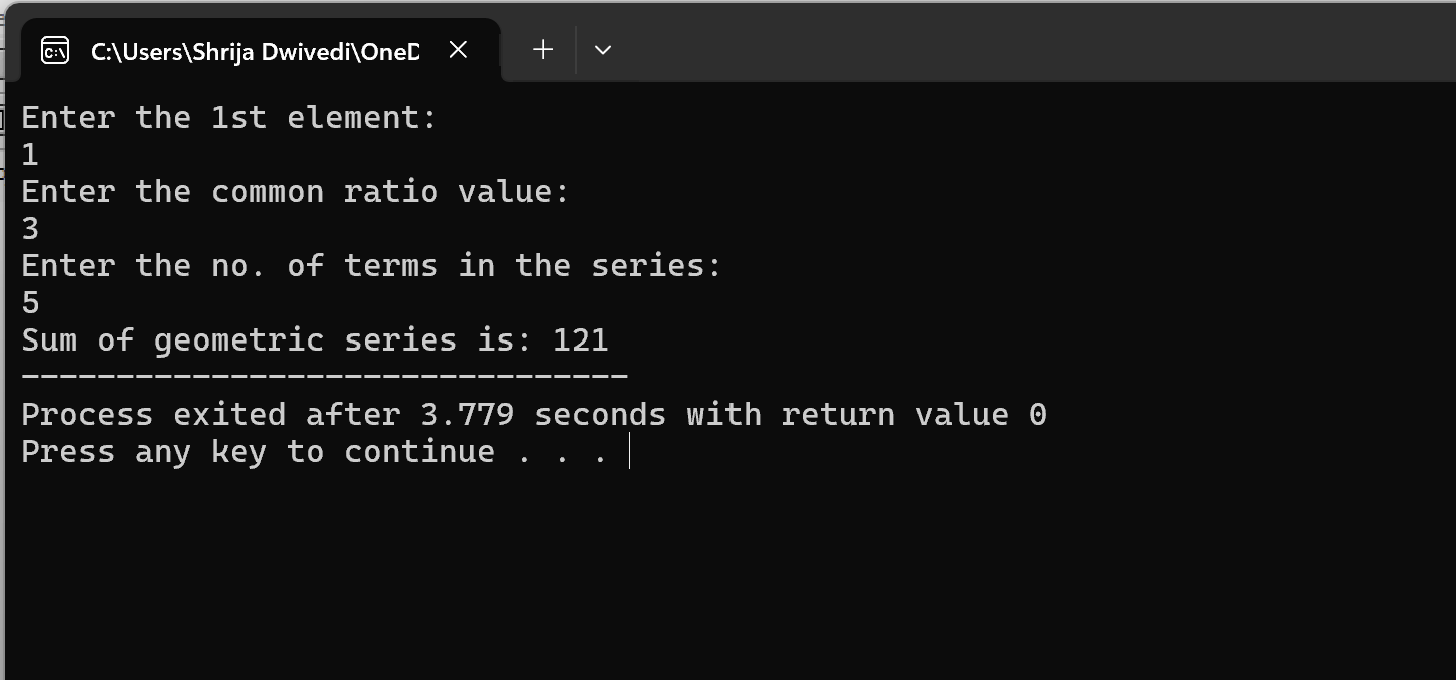
sum=(a\*pow(r,n)-1)/(r-1);

printf("Sum of geometric series is: %d", sum);

return 0;

}

Output:



Method 2:

#include<stdio.h>

int main(){

int a,n,r,i;

printf("Enter the 1st element:\n");

scanf("%d", &a);

printf("Enter the common ratio value:\n");

scanf("%d", &r);

printf("Enter the no. of terms in the series:\n");

scanf("%d", &n);

int sum=0;

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

sum=sum+a;

a=a\*r;

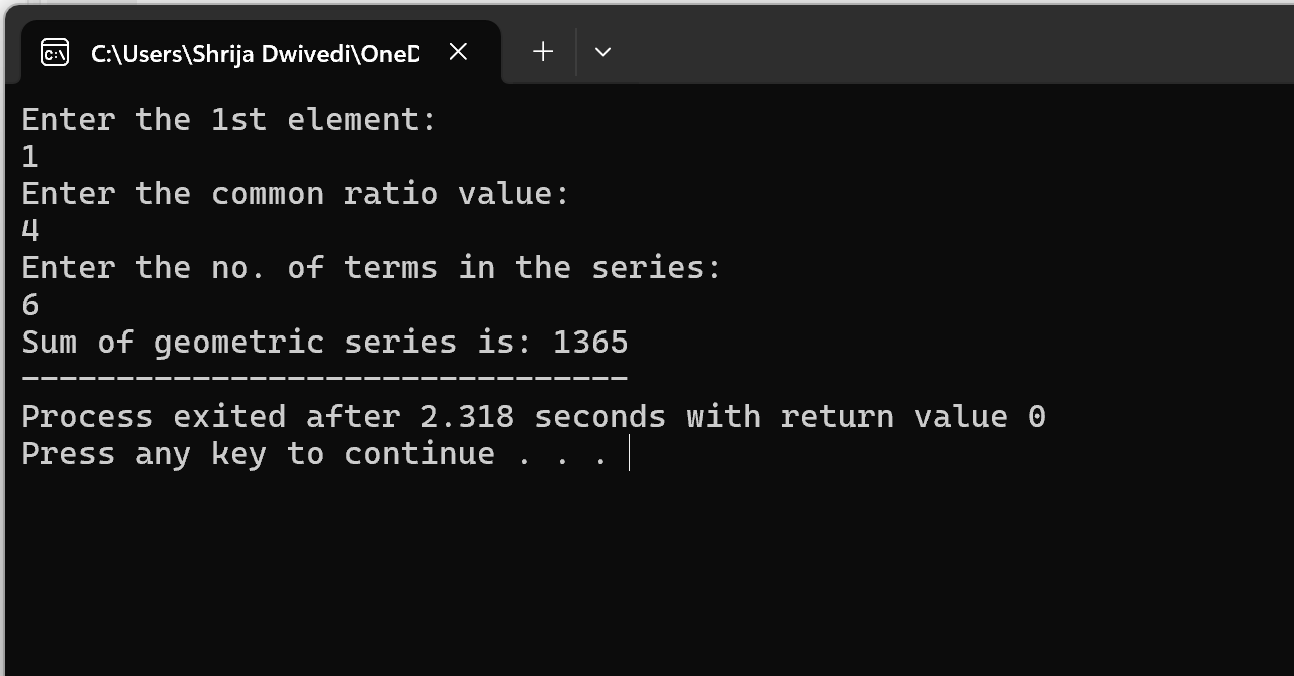
}

printf("Sum of geometric series is: %d", sum);

return 0;

}

output:



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

#include <stdio.h>

#include <stdlib.h>

struct newnode {

int data;

struct newnode \*left, \*right;

};

struct newnode\* createbst(int value)

{

struct newnode\* temp= (struct newnode\*)malloc(sizeof(struct newnode));

temp->data = value;

temp->left = NULL;

temp->right = NULL;

return temp;

}

struct newnode\* insertbst(struct newnode\* root, int value)

{

if (root == NULL) {

return createbst(value);

}

if (value < root->data) {

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

}

else if (value > root->data) {

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

}

return root;

}

void inorder(struct newnode\* root)

{

if (root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

}

struct newnode\* Min(struct newnode\* root)// Function to find the minimum value

{

if (root == NULL) {

return NULL;

}

else if (root->left != NULL) {

return Min(root->left);

}

return root;

}

struct newnode\* deletebst (struct newnode\* root, int key)

{

if (root == NULL)

return NULL;

if (key > root->data) {

root->right = deletebst (root->right, key);

}

else if (key < root->data) {

root->left = deletebst (root->left, key);

}

else {

if (root->left == NULL && root->right == NULL) {

free(root);

return NULL;

}

else if (root->left == NULL || root->right == NULL) {

struct newnode\* temp;

if (root->left == NULL) {

temp = root->right;

}

else {

temp = root->left;

}

free(root);

return temp;

}

else {

struct newnode\* temp

= Min(root->right);

root->data = temp->data;

root->right = deletebst (root->right, temp->data);

}

}

return root;

}

int main()

{

struct newnode\* root = NULL;

int arr[7],n,i,a;

printf("Enter 7 numbers to insert \n");

for ( i=0;i<7;i++){

scanf("%d",&arr[i]);

}

for (i=0;i<7;i++){

root = insertbst(root,arr[i]);

}

printf("Inorder Traversal Before Deletion \n");

inorder(root);

printf("\n");

printf("Enter a number to be deleted \n");

scanf("%d",&a);

root = deletebst (root,a);

printf("Inorder Traversal After Deletion: \n");

inorder(root);

return 0;

}

Output

