**LAB EXPERIMENTS**

1. **Write a program to print Fibonacci series using recursion.**

**Program:**

#include <stdio.h>

int fibonacci(int n) {

if (n == 0 || n == 1) {

return n;

} else

{

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

int main()

{

int n,i;

printf("Enter number of terms: ");

scanf("%d", &n);

printf("Fibonacci Series: ");

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

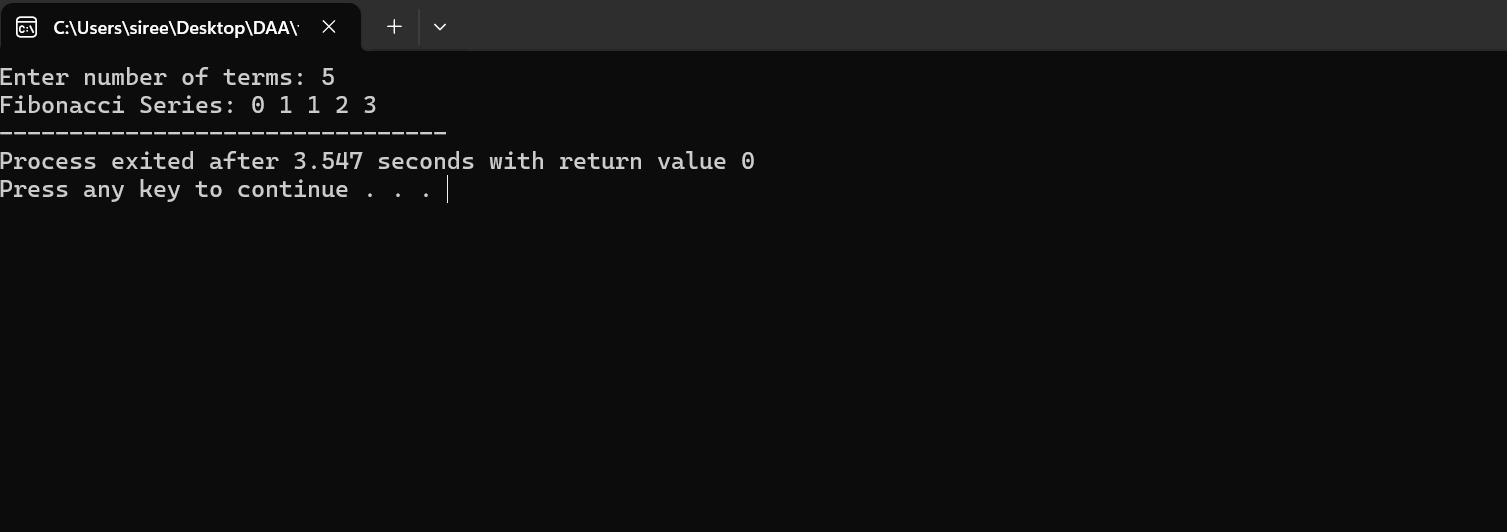
printf("%d ", fibonacci(i));

}

return 0;

}

**Output**

****

**2.Write a program to check the given number is Armstrong or not**

**Program:**

#include <stdio.h>

int main() {

int num, sum = 0, rem, temp, digits;

printf("Enter a number: ");

scanf("%d", &num);

temp = num;

while (temp != 0) {

rem = temp % 10;

sum += rem \* rem \* rem;

temp /= 10;

}

if (sum == num) {

printf("%d is an Armstrong number.", num);

} else {

printf("%d is not an Armstrong number.", num);

}

return 0;

}

**Output:**

****

**3.Write a program to find the gcd of two numbers.**

**Program:**

#include <stdio.h>

int gcd(int a, int b) {

if (b == 0) {

return a;

} else {

return gcd(b, a % b);

}

}

int main() {

int num1, num2;

printf("Enter first number: ");

scanf("%d", &num1);

printf("Enter second number: ");

scanf("%d", &num2);

printf("GCD of %d and %d is: %d\n", num1, num2, gcd(num1, num2));

return 0;

}

**Output:**

****

**4.Write a program to get the largest element of an array.**

**Program:**

#include <stdio.h>

int main() {

int n;

printf("Enter number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter array elements:\n");

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

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

}

int max = arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] > max) {

max = arr[i];

}

}

printf("Largest element: %d\n", max);

return 0;

}

**Output:**

****

**5.write a program to find the factorial of a number.**

**Program:**

#include <stdio.h>

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

int fact=1,i;

for (i = 1; i <= num; i++) {

fact\*= i;

}

printf("Factorial of %d is: %d\n", num, fact);

return 0;

}

**Output:**

****

**6.write a program to check a number is a prime number or not.**

**Program:**

#include <stdio.h>

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

int isPrime = 1;

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) {

isPrime = 0;

break;

}

}

if (num <= 1) {

isPrime = 0;

}

if (isPrime) {

printf("%d is a prime number.\n", num);

} else {

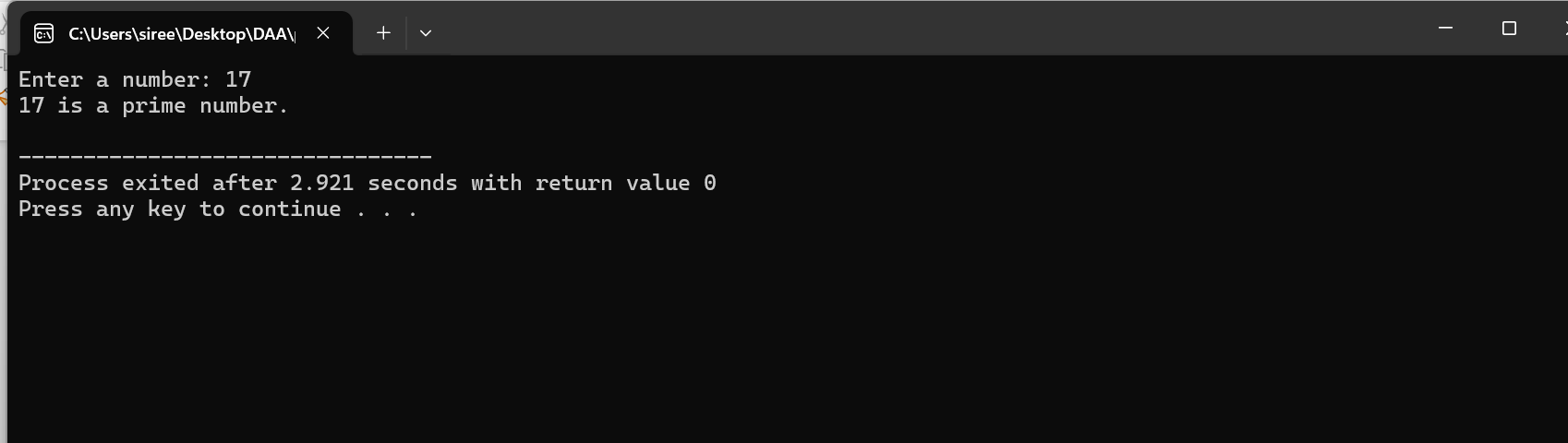
printf("%d is not a prime number.\n", num);

}

return 0;

}

**Output:**



7. **Write a program to perform selection sort.**

**Program:**

}#include <stdio.h>

int main() {

int n, i, j, min, temp;

printf("Enter number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter array elements:\n");

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

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

}

for (i = 0; i < n - 1; i++) {

min = i;

for (j = i + 1; j < n; j++) {

if (arr[j] < arr[min]) {

min = j;

}

}

if (min != i) {

temp = arr[i];

arr[i] = arr[min];

arr[min] = temp;

}

}

printf("Sorted array: ");

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

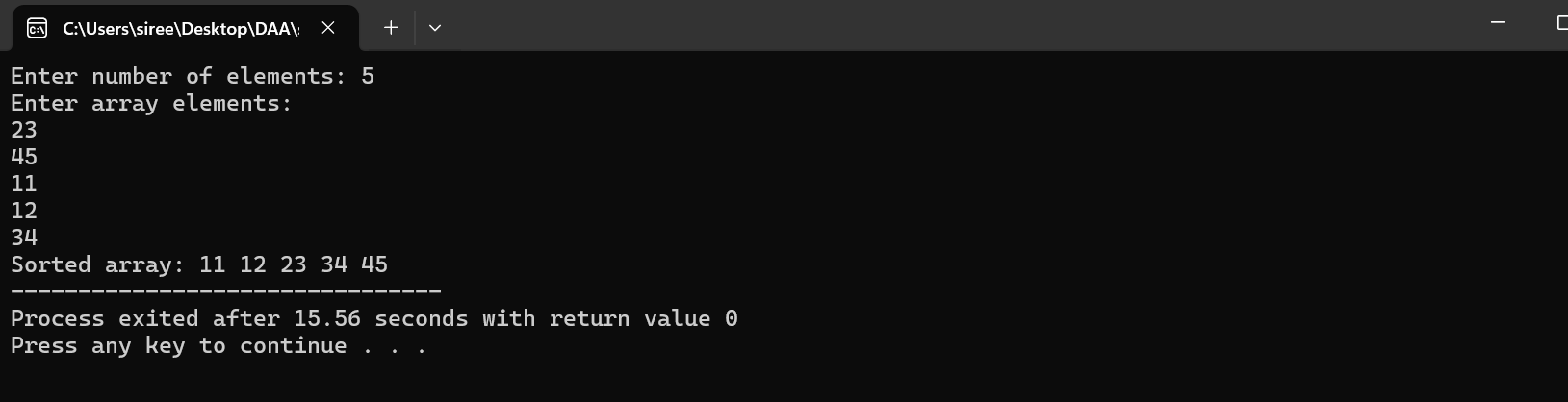
printf("%d ", arr[i]);

}

return 0;

}

**Output:**

****

**8.write a program to perform bubble sort**

**Program:**

#include <stdio.h>

int main() {

int n, i, j, temp;

printf("Enter number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter array elements:\n");

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

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

}

for (i = 0; i < n - 1; i++) {

for (j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

printf("Sorted array: ");

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

printf("%d ", arr[i]);

}

return 0;

}

**Output:**

****

**9. write a program to multiply two matrices**

**Program:**

#include <stdio.h>

int main() {

int m1[2][2], m2[2][2], result[2][2];

int i, j, k;

printf("Enter Matrix 1 elements:\n");

for (i = 0; i < 2; i++) {

for (j = 0; j < 2; j++) {

scanf("%d", &m1[i][j]);

}

}

printf("Enter Matrix 2 elements:\n");

for (i = 0; i < 2; i++) {

for (j = 0; j < 2; j++) {

scanf("%d", &m2[i][j]);

}

}

for (i = 0; i < 2; i++) {

for (j = 0; j < 2; j++) {

result[i][j] = 0;

for (k = 0; k < 2; k++) {

result[i][j] += m1[i][k] \* m2[k][j];

}

}

}

printf("Result:\n");

for (i = 0; i < 2; i++) {

for (j = 0; j < 2; j++) {

printf("%d ", result[i][j]);

}

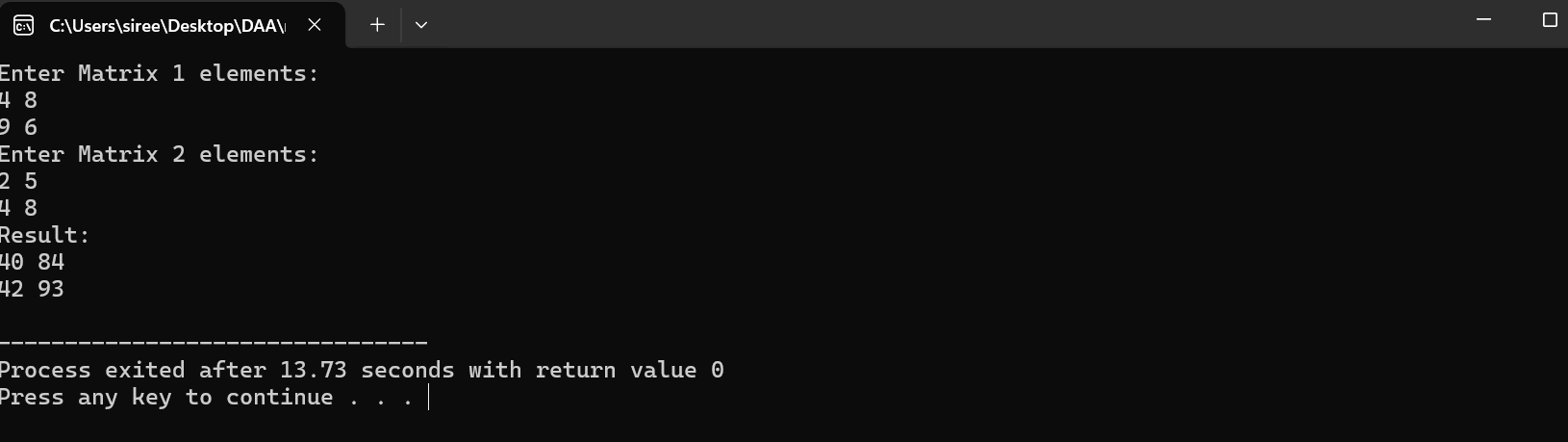
printf("\n");

}

return 0;

}

**Output:**

****

**10.write a program to check whether the given string is palindrome or not**

**Program:**

#include <stdio.h>

#include <string.h>

int main() {

char str[100];

int i, len;

printf("Enter a string: ");

scanf("%s", str);

len = strlen(str);

int flag = 0;

for (i = 0; i < len / 2; i++) {

if (str[i] != str[len - i - 1]) {

flag = 1;

break;

}

}

if (flag == 0) {

printf("%s is a palindrome.\n", str);

} else {

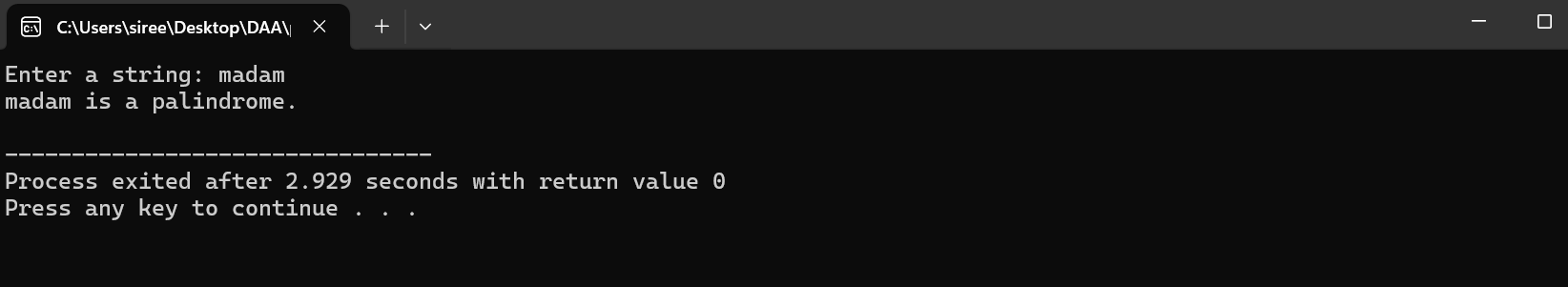
printf("%s is not a palindrome.\n", str);

}

return 0;

}

**Output:**

****

**11.write a program for to copy one string to another**

**Program:**

#include <stdio.h>

int main() {

char src[100], dest[100];

printf("Enter source string: ");

scanf("%s", src);

int i;

for (i = 0; src[i] != '\0'; i++) {

dest[i] = src[i];

}

dest[i] = '\0';

printf("Destination string: %s\n", dest);

return 0;

}

**Output:**

****

**12.write a program to perform binary search**

**Program:**

#include <stdio.h>

int main() {

int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};

int n = 10;

int target, left = 0, right = n - 1;

printf("Enter target value: ");

scanf("%d", &target);

while (left <= right) {

int mid = (left + right) / 2;

if (arr[mid] == target) {

printf("Target found at index %d.\n", mid);

return 0;

} else if (arr[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

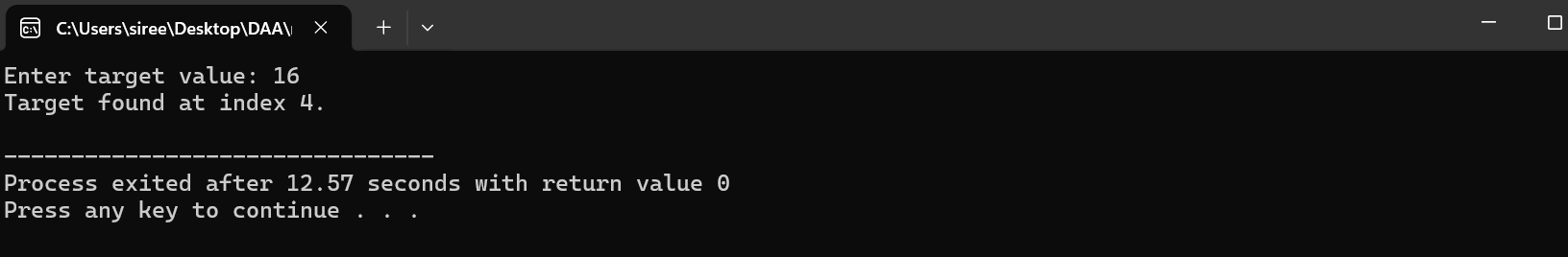
}

printf("Target not found in array.\n");

return 0;

}

**Output:**

****

**13.write a program to print the reverse of a string**

**Program:**

#include <stdio.h>

#include <string.h>

int main() {

char str[100];

printf("Enter a string: ");

scanf("%s", str);

int len = strlen(str);

printf("Reverse: ");

for (int i = len - 1; i >= 0; i--) {

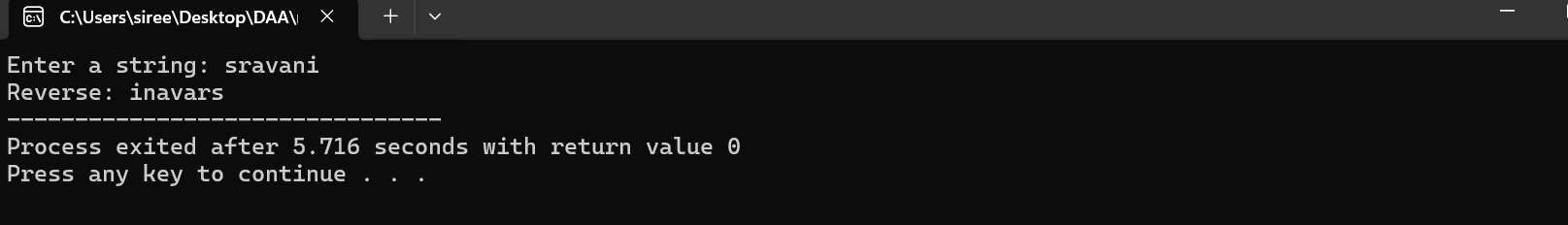
printf("%c", str[i]);

}

return 0;

}

**Output:**

****

**14.write a program to find the length of a string.**

**Program:**

#include <stdio.h>

int main() {

char str[100];

printf("Enter a string: ");

scanf("%s", str);

int len = 0;

while (str[len] != '\0') {

len++;

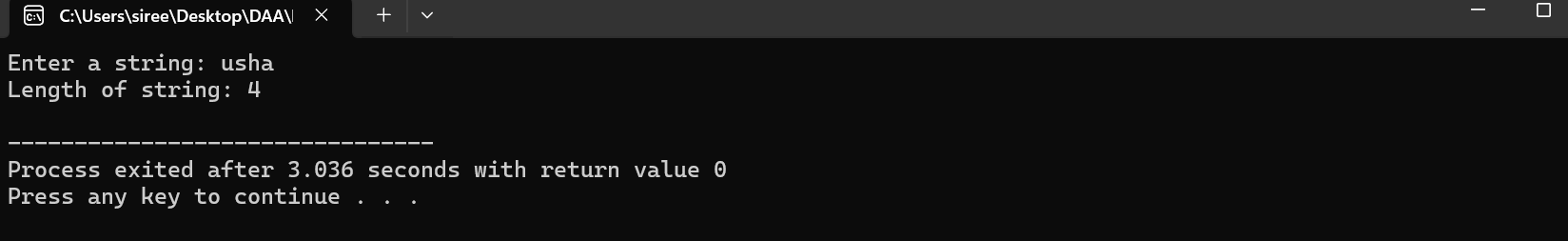
}

printf("Length of string: %d\n", len);

return 0;

}

**Output:**

****

**15.Write a program to perform Strassen’s matrix multiplication.**

**Program:**

#include <stdio.h>

#include <stdlib.h>

void strassen(int\*\* A, int\*\* B, int\*\* C) {

int M1 = (A[0][0] + A[1][1]) \* (B[0][0] + B[1][1]);

int M2 = (A[1][0] + A[1][1]) \* B[0][0];

int M3 = A[0][0] \* (B[0][1] - B[1][1]);

int M4 = A[1][1] \* (B[1][0] - B[0][0]);

int M5 = (A[0][0] + A[0][1]) \* B[1][1];

int M6 = (A[1][0] - A[0][0]) \* (B[0][0] + B[0][1]);

int M7 = (A[0][1] - A[1][1]) \* (B[1][0] + B[1][1]);

C[0][0] = M1 + M4 - M5 + M7;

C[0][1] = M3 + M5;

C[1][0] = M2 + M4;

C[1][1] = M1 - M2 + M3 + M6;

}

int main() {

int\*\* A = (int\*\*)malloc(2 \* sizeof(int\*));

int\*\* B = (int\*\*)malloc(2 \* sizeof(int\*));

int\*\* C = (int\*\*)malloc(2 \* sizeof(int\*));

for (int i = 0; i < 2; i++) {

A[i] = (int\*)malloc(2 \* sizeof(int));

B[i] = (int\*)malloc(2 \* sizeof(int));

C[i] = (int\*)malloc(2 \* sizeof(int));

}

A[0][0] = 1; A[0][1] = 2;

A[1][0] = 3; A[1][1] = 4;

B[0][0] = 5; B[0][1] = 6;

B[1][0] = 7; B[1][1] = 8;

strassen(A, B, C);

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

printf("%d ", C[i][j]);

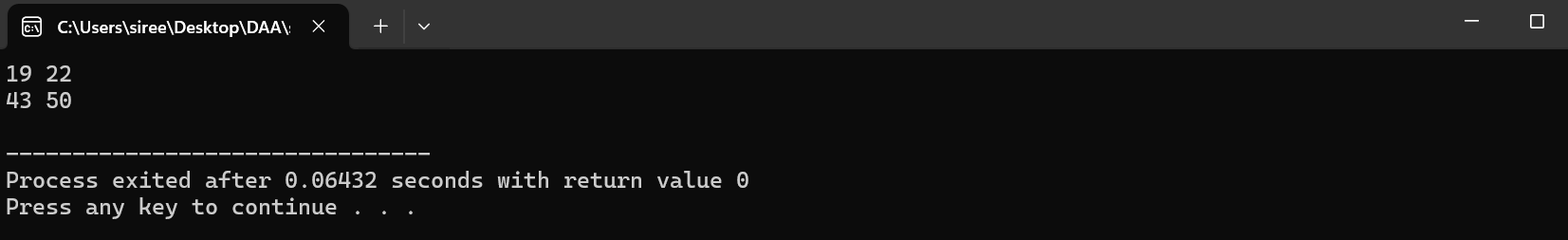
}

printf("\n");

}

}

**Output:**



**16.Write a program to perform merge sort.**

**Program:**

#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {

int i, j, k;

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

for (i = 0; i < n1; i++) L[i] = arr[left + i];

for (j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

i = 0; j = 0; k = left;

while (i < n1 && j < n2) {

arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

}

while (i < n1) arr[k++] = L[i++];

while (j < n2) arr[k++] = R[j++];

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

int main() {

int arr[] = {12, 11, 13, 5, 6, 7};

int n = sizeof(arr) / sizeof(arr[0]);

mergeSort(arr, 0, n - 1);

printf("Sorted array: ");

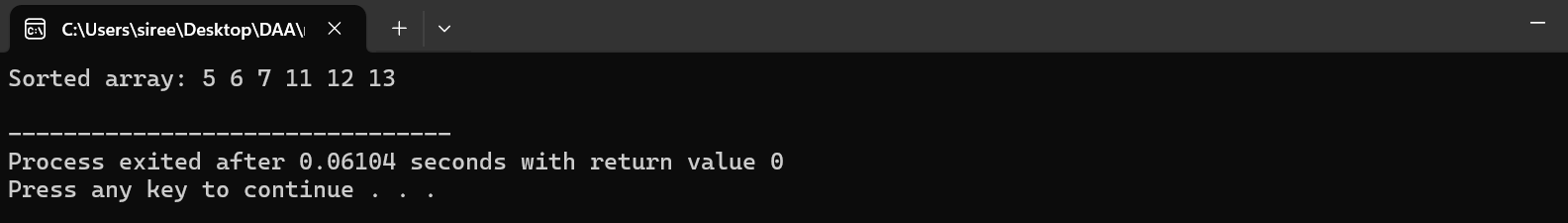
for (int i = 0; i < n; i++) printf("%d ", arr[i]);

printf("\n");

return 0;

}

**Output:**



**17.Using divide and conquer strategy find max and min in the list.**

**Program:**

#include <stdio.h>

void maxMin(int arr[], int left, int right, int\* max, int\* min) {

if (left == right) {

\*max = \*min = arr[left];

} else {

int mid = (left + right) / 2;

int max1, min1, max2, min2;

maxMin(arr, left, mid, &max1, &min1);

maxMin(arr, mid + 1, right, &max2, &min2);

\*max = (max1 > max2) ? max1 : max2;

\*min = (min1 < min2) ? min1 : min2;

}

}

int main() {

int arr[] = {5, 2, 8, 1, 9};

int n = sizeof(arr) / sizeof(arr[0]);

int max, min;

maxMin(arr, 0, n - 1, &max, &min);

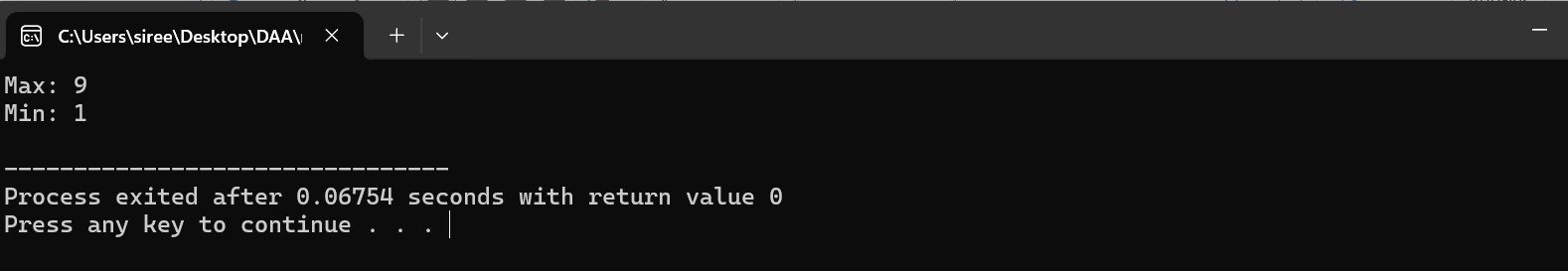
printf("Max: %d\n", max);

printf("Min: %d\n", min);

return 0;

}

**Output:**



**18.write a program to generate all primes in given range.**

**Program:**

#include <stdio.h>

int main() {

int lower, upper;

printf("Enter lower bound: ");

scanf("%d", &lower);

printf("Enter upper bound: ");

scanf("%d", &upper);

printf("Prime numbers between %d and %d:\n", lower, upper);

for (int i = lower; i <= upper; i++) {

int isPrime = 1;

if (i < 2) continue;

for (int j = 2; j \* j <= i; j++) {

if (i % j == 0) {

isPrime = 0;

break;

}

}

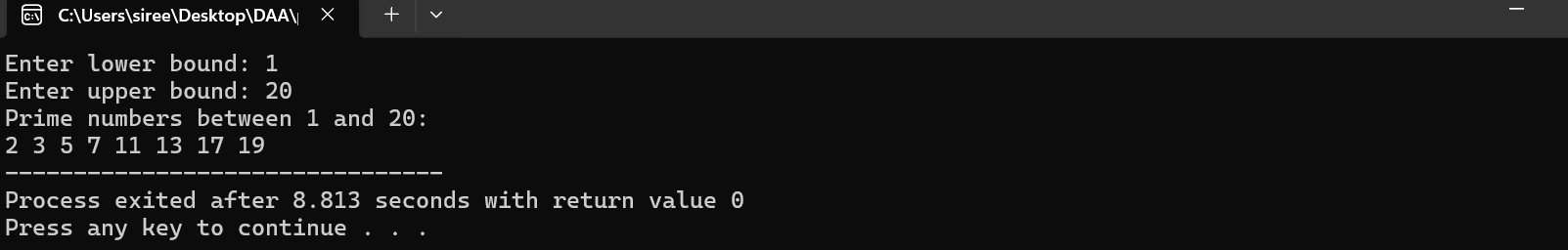
if (isPrime) printf("%d ", i);

}

return 0;

}

**Output:**



**19.Write a program to perform knapsack problem using greedy technique.**

**Program:**

#include <stdio.h>

int main() {

int n, capacity;

printf("Enter number of items: ");

scanf("%d", &n);

printf("Enter knapsack capacity: ");

scanf("%d", &capacity);

int weight[n], value[n], ratio[n];

printf("Enter weights and values:\n");

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

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

ratio[i] = value[i] / weight[i];

}

for (int i = 0; i < n - 1; i++) {

for (int j = i + 1; j < n; j++) {

if (ratio[i] < ratio[j]) {

int temp = weight[i];

weight[i] = weight[j];

weight[j] = temp;

temp = value[i];

value[i] = value[j];

value[j] = temp;

temp = ratio[i];

ratio[i] = ratio[j];

ratio[j] = temp;

}

}

}

int totalValue = 0;

printf("Selected items:\n");

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

if (weight[i] <= capacity) {

capacity -= weight[i];

totalValue += value[i];

printf("Weight: %d, Value: %d\n", weight[i], value[i]);

} else {

int fraction = capacity;

totalValue += fraction \* ratio[i];

printf("Weight: %d, Value: %d\n", fraction, fraction \* ratio[i]);

break;

}

}

printf("Total value: %d\n", totalValue);

return 0;

}

**Output:**

****

**20.write a program to perform MST using greedy techniques**

**Program:**

#include <stdio.h>

#define V 4

#define E 5

void kruskal(int edge[E][3]) {

int parent[V];

int rank[V];

int result[E][3];

int i, e = 0;

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

for (int j = i + 1; j < E; j++) {

if (edge[i][2] > edge[j][2]) {

int temp[3];

temp[0] = edge[i][0];

temp[1] = edge[i][1];

temp[2] = edge[i][2];

edge[i][0] = edge[j][0];

edge[i][1] = edge[j][1];

edge[i][2] = edge[j][2];

edge[j][0] = temp[0];

edge[j][1] = temp[1];

edge[j][2] = temp[2];

}

}

}

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

parent[i] = i;

rank[i] = 0;

}

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

int u = edge[i][0];

int v = edge[i][1];

if (parent[u] != parent[v]) {

result[e][0] = u;

result[e][1] = v;

result[e][2] = edge[i][2];

e++;

int temp = parent[u];

parent[v] = temp;

}

}

printf("Edges in MST:\n");

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

printf("%d -- %d == %d\n", result[i][0], result[i][1], result[i][2]);

}

}

int main() {

int edge[E][3] = {{0, 1, 10}, {0, 2, 6}, {0, 3, 5}, {1, 3, 15}, {2, 3, 4}};

kruskal(edge);

return 0;

}

**Output:**

****

**21.using dynamic programming concept to find out optimal binary search tree**

**Program:**

#include <stdio.h>

#define N 5

#define INF 9999

int keys[N] = {10, 20, 30, 40, 50};

int freq[N] = {34, 23, 17, 14, 12};

int sum[N + 1] = {0};

int dp[N + 1][N + 1];

void optimalBST() {

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

sum[i] = (i == 0) ? 0 : sum[i - 1] + freq[i - 1];

}

for (int l = 1; l <= N; l++) {

for (int i = 0; i <= N - l; i++) {

int j = i + l - 1;

dp[i][j] = INF;

for (int r = i; r <= j; r++) {

int cost = sum[j + 1] - sum[i] + (r == i ? 0 : dp[i][r - 1]) + (r == j ? 0 : dp[r + 1][j]);

dp[i][j] = (cost < dp[i][j]) ? cost : dp[i][j];

}

}

}

printf("Optimal BST Cost: %d\n", dp[0][N - 1]);

}

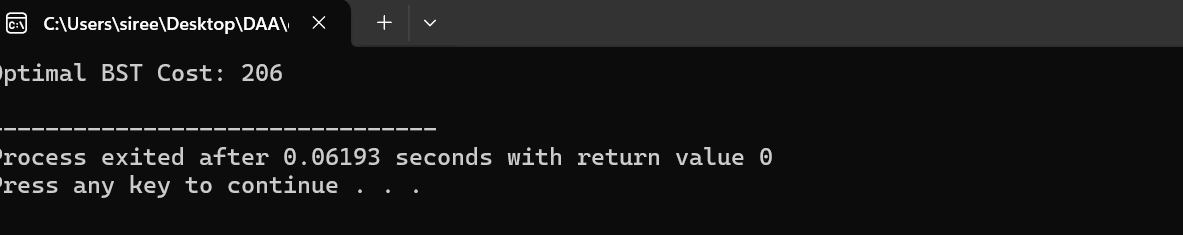
int main() {

optimalBST();

return 0;

}

**Output:**

****

**22.** **use dynamic programming to find binomial coefficient of a given number**

**Program:**

#include <stdio.h>

int binomialCoefficient(int n, int k) {

int C[n + 1][k + 1];

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

C[i][0] = 1;

}

for (int j = 0; j <= k; j++) {

C[j][j] = 1;

}

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= k && j <= i; j++) {

C[i][j] = C[i - 1][j - 1] + C[i - 1][j];

}

}

return C[n][k];

}

int main() {

int n, k;

printf("Enter value of n: ");

scanf("%d", &n);

printf("Enter value of k: ");

scanf("%d", &k);

if (k > n) {

printf("Error: k cannot be greater than n");

} else {

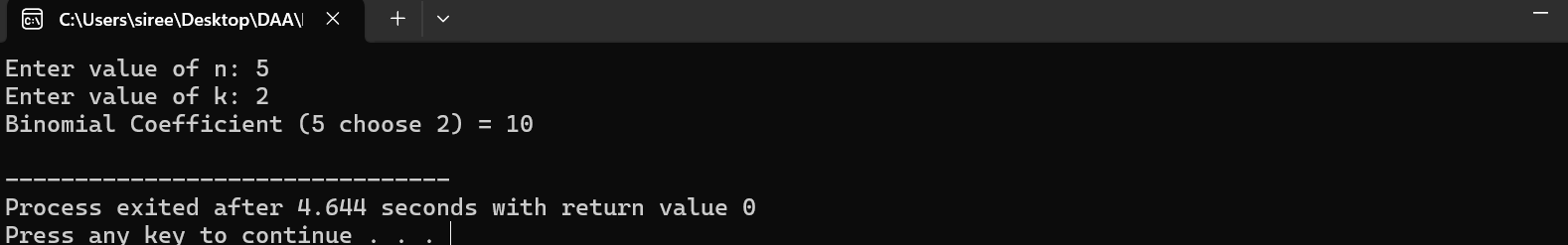
printf("Binomial Coefficient (%d choose %d) = %d\n", n, k, binomialCoefficient(n, k));

}

return 0;

**}**

**Output:**



**23.Write a program to reverse a given number.**

**Program:**

#include <stdio.h>

int main() {

int num, reverse = 0;

printf("Enter a number: ");

scanf("%d", &num);

while (num != 0) {

reverse = reverse \* 10 + num % 10;

num /= 10;

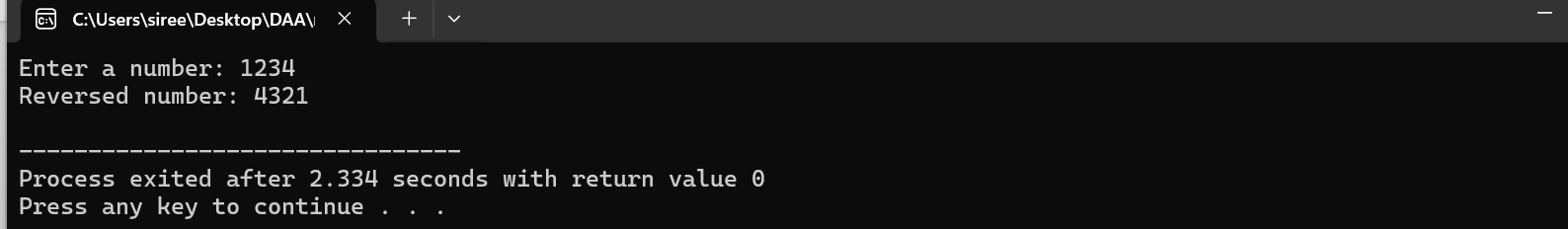
}

printf("Reversed number: %d\n", reverse);

return 0;

}

**Output:**

****

**24.Write a program to check whether given number is perfect number or not**

**Program:**

#include <stdio.h>

int main() {

int num, sum = 0;

printf("Enter a number: ");

scanf("%d", &num);

for (int i = 1; i < num; i++) {

if (num % i == 0) {

sum += i;

}

}

if (sum == num) {

printf("%d is a perfect number\n", num);

} else {

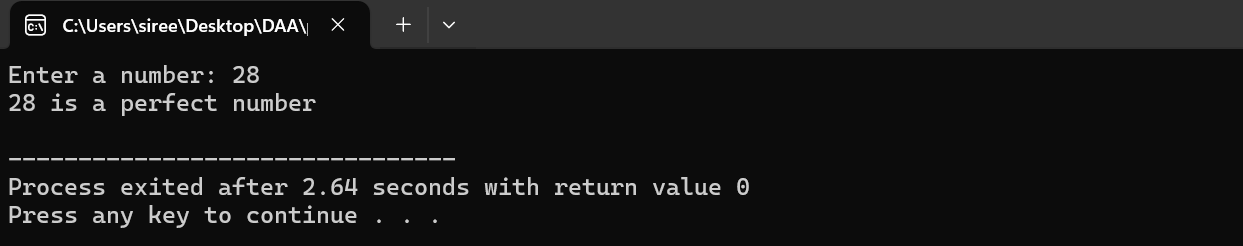
printf("%d is not a perfect number\n", num);

}

return 0;

}

**Output:**

****

**25.Write a program to perform travelling salesman problem using dynamic programming**

**Program:**

#include <stdio.h>

#include <limits.h>

#define MAX 20

#define INF INT\_MAX

int dist[MAX][MAX];

int memo[MAX][1 << MAX];

int tsp(int city, int mask, int n) {

if (mask == (1 << n) - 1) {

return dist[city][0];

}

if (memo[city][mask] != -1) {

return memo[city][mask];

}

int res = INF;

for (int next\_city = 0; next\_city < n; next\_city++) {

if (!(mask & (1 << next\_city))) {

int new\_cost = dist[city][next\_city] + tsp(next\_city, mask | (1 << next\_city), n);

res = (res < new\_cost) ? res : new\_cost;

}

}

memo[city][mask] = res;

return res;

}

int main() {

int n;

printf("Enter the number of cities: ");

scanf("%d", &n);

printf("Enter the distance matrix:\n");

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

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

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

}

}

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

for (int j = 0; j < (1 << n); j++) {

memo[i][j] = -1;

}

}

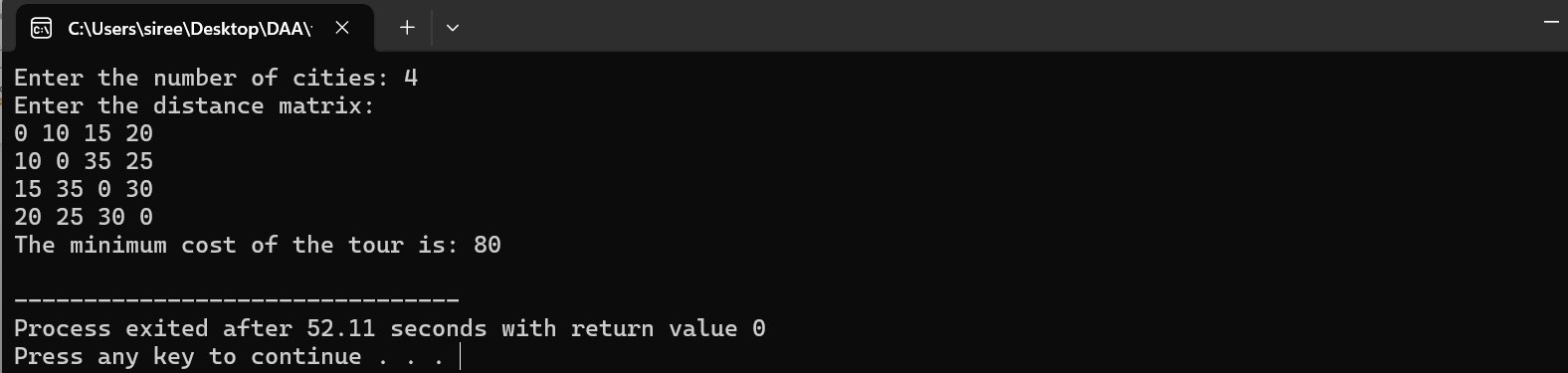
int min\_cost = tsp(0, 1, n);

printf("The minimum cost of the tour is: %d\n", min\_cost);

return 0;

}

Output:



**26.write a program for number pyramid**

**Program:**

#include <stdio.h>

int main() {

int rows;

printf("Enter the number of rows: ");

scanf("%d", &rows);

for (int i = 1; i <= rows; i++) {

for (int j = 1; j <= rows - i; j++) {

printf(" ");

}

for (int k = 1; k <= i; k++) {

printf("%d ", k);

}

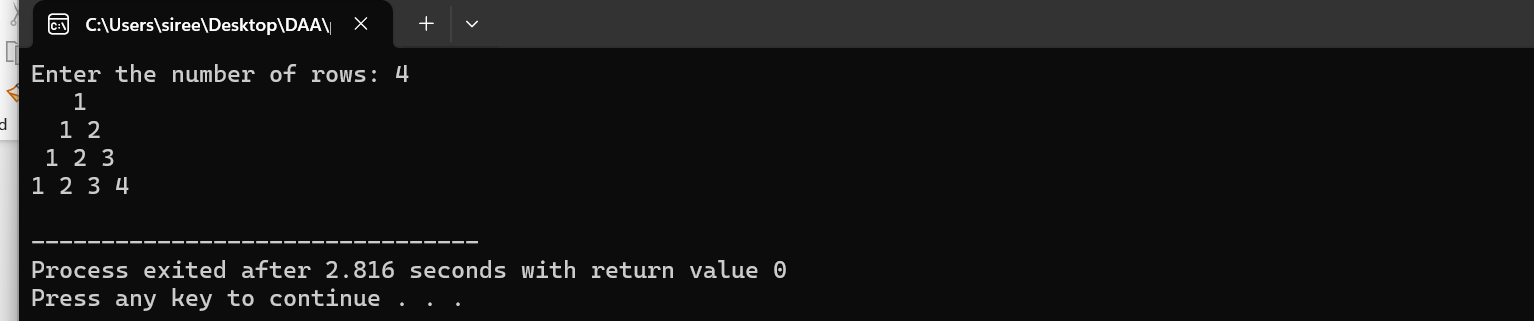
printf("\n");

}

return 0;

}

Output:



**27.Write a program to perform Floyd’s algorithm.**

**Program:**

#include <stdio.h>

#include <limits.h>

#define INF 9999

#define V 4

void floydWarshall(int graph[V][V]) {

for (int k = 0; k < V; k++) {

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

for (int j = 0; j < V; j++) {

if (graph[i][k] + graph[k][j] < graph[i][j]) {

graph[i][j] = graph[i][k] + graph[k][j];

}

}

}

}

printf("Shortest Distances:\n");

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

for (int j = 0; j < V; j++) {

if (graph[i][j] == INF) {

printf("%s ", "INF");

} else {

printf("%d ", graph[i][j]);

}

}

printf("\n");

}

}

int main() {

int graph[V][V] = {

{0, 5, INF, 10},

{INF, 0, 3, INF},

{INF, INF, 0, 1},

{INF, INF, INF, 0}

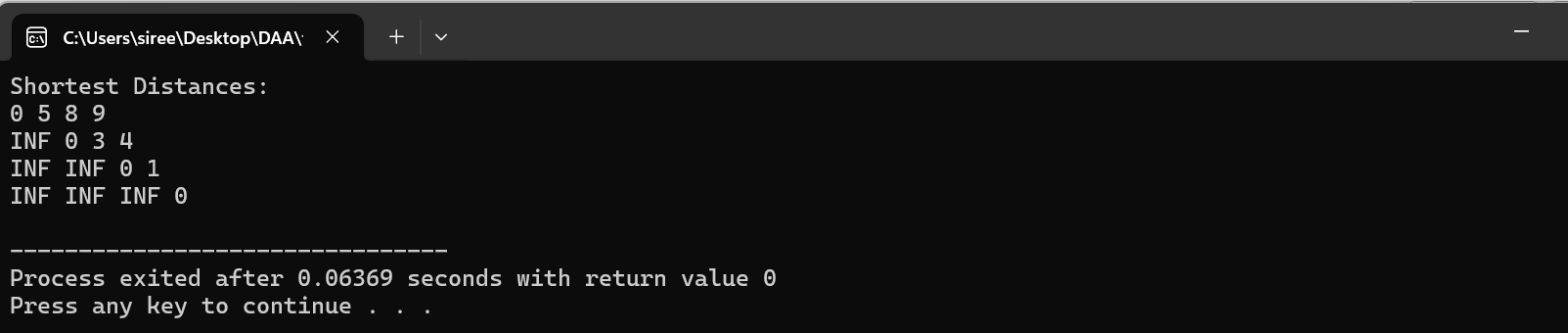
};

floydWarshall(graph);

return 0;

}

**Output:**



**28.Write a program for pascal triangle pattern.**

**Program:**

#include <stdio.h>

int main() {

int rows;

printf("Enter the number of rows: ");

scanf("%d", &rows);

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

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

int res = 1;

if (j > 0 && i - j > 0) {

res = (i \* (i - 1)) / (j \* (i - j));

}

printf("%d ", res);

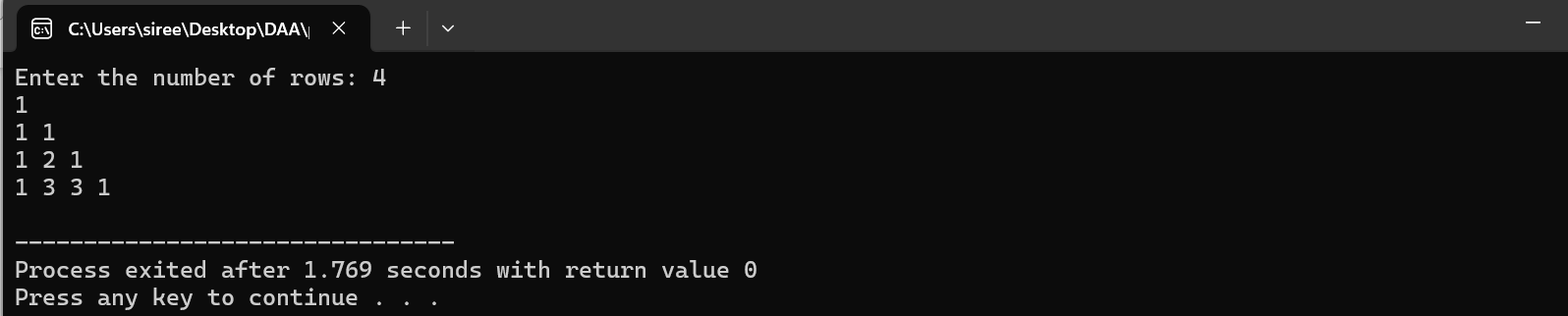
}

printf("\n");

}

}

**Output:**

****

**29.Write a program to find the optimal cost by using appropriate algorithm.**

**Program:**

#include <stdio.h>

int main() {

int N, W;

printf("Enter number of items: ");

scanf("%d", &N);

printf("Enter knapsack capacity: ");

scanf("%d", &W);

int weights[N], values[N];

printf("Enter weights and values:\n");

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

printf("Item %d: Weight = ", i + 1);

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

printf("Value = ");

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

}

int dp[N + 1][W + 1];

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

for (int j = 0; j <= W; j++) {

if (i == 0 || j == 0) {

dp[i][j] = 0;

}

}

}

for (int i = 1; i <= N; i++) {

for (int j = 1; j <= W; j++) {

if (weights[i - 1] <= j) {

dp[i][j] = (values[i - 1] + dp[i - 1][j - weights[i - 1]] > dp[i - 1][j]) ?

values[i - 1] + dp[i - 1][j - weights[i - 1]] : dp[i - 1][j];

} else {

dp[i][j] = dp[i - 1][j];

}

}

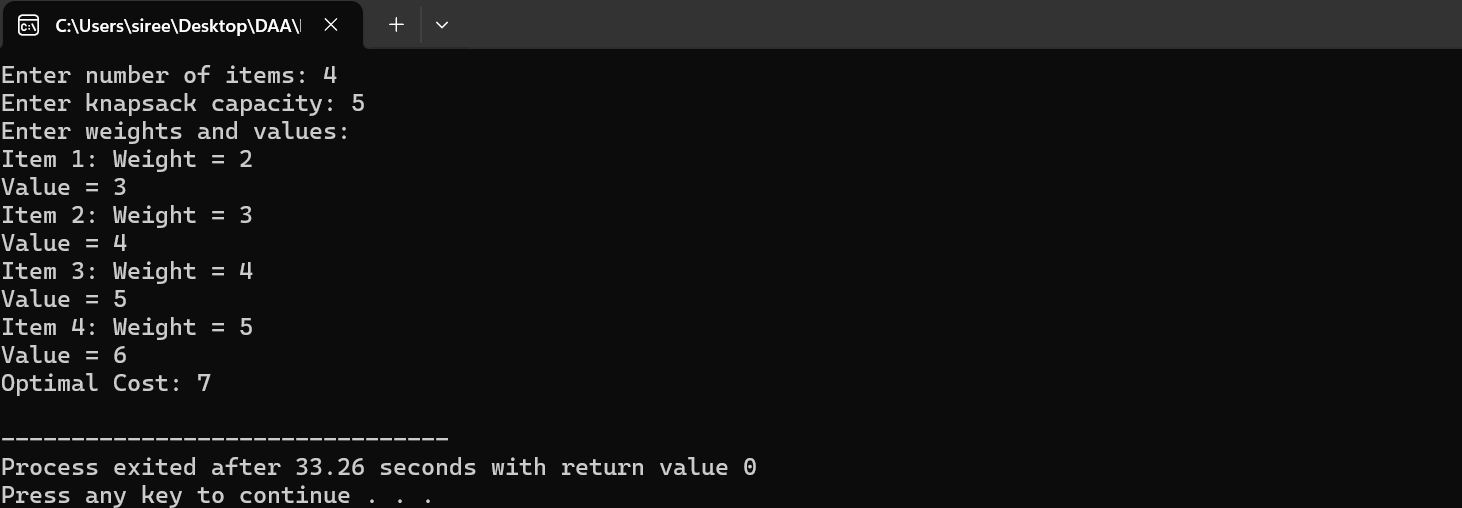
}

printf("Optimal Cost: %d\n", dp[N][W]);

return 0;

}

**Output:**



**30.Write a program to find the sum of the digits.**

**Program**:

#include <stdio.h>

int main() {

int num, sum = 0, digit;

printf("Enter a number: ");

scanf("%d", &num);

while (num != 0) {

digit = num % 10;

sum += digit;

num /= 10;

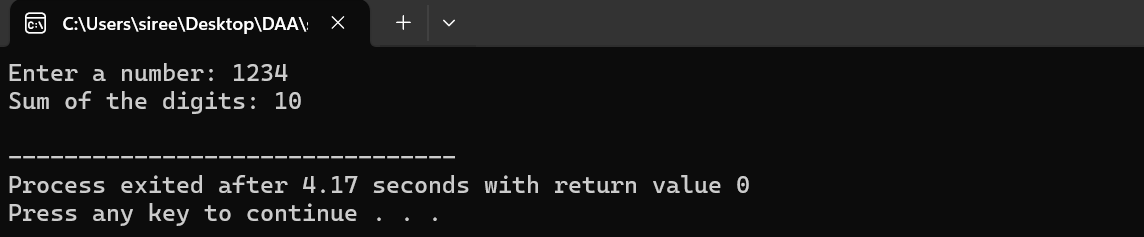
}

printf("Sum of the digits: %d\n", sum);

return 0;

}

**Output:**

****

**31.Write a program to print a maximum and minimum value sequency for all numbers in a list.**

**Program:**

#include <stdio.h>

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d numbers: ", n);

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

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

}

int max\_val = arr[0], min\_val = arr[0];

printf("Input List: ");

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

printf("%d ", arr[i]);

if (arr[i] > max\_val) {

max\_val = arr[i];

}

if (arr[i] < min\_val) {

min\_val = arr[i];

}

}

printf("\n");

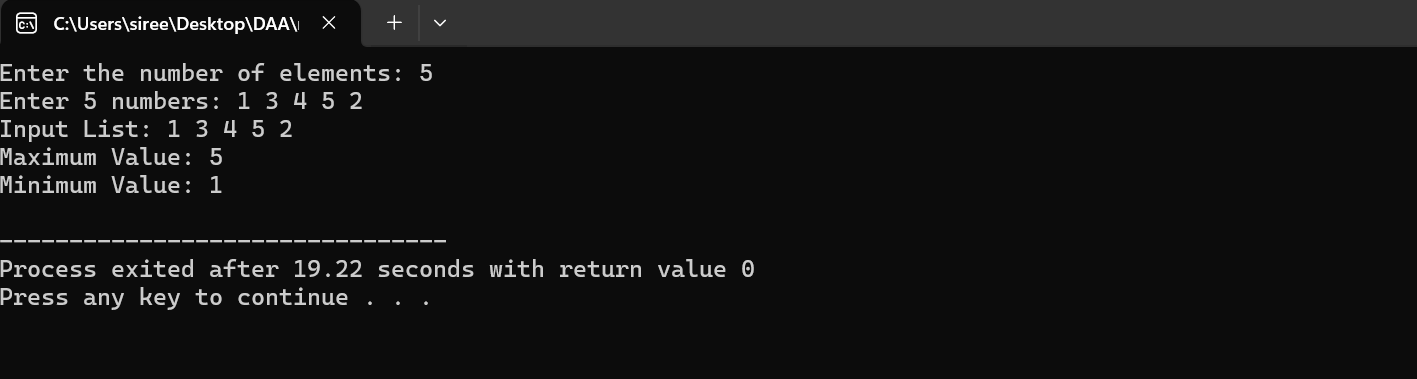
printf("Maximum Value: %d\n", max\_val);

printf("Minimum Value: %d\n", min\_val);

return 0;

}

**Output:**



**32.write a program to perform n queen problem by using backtracking.**

**Program:**

#include <stdio.h>

#define N 4

int board[N][N];

int isSafe(int row, int col) {

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

if (board[row][i] == 1) return 0;

}

for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {

if (board[i][j] == 1) return 0;

}

for (int i = row, j = col; j >= 0 && i < N; i++, j--) {

if (board[i][j] == 1) return 0;

}

return 1;

}int solve(int col) {

if (col == N) return 1;

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

if (isSafe(i, col)) {

board[i][col] = 1;

if (solve(col + 1)) ret

board[i][col] = 0;

}

}

return 0;

}

void print() {

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

for (int j = 0; j < N; j++) {

printf("%d ", board[i][j]);

}

printf("\n");

}

}

int main() {

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

for (int j = 0; j < N; j++) {

board[i][j] = 0;

}

}

if (solve(0)) {

printf("Solution found:\n");

print();

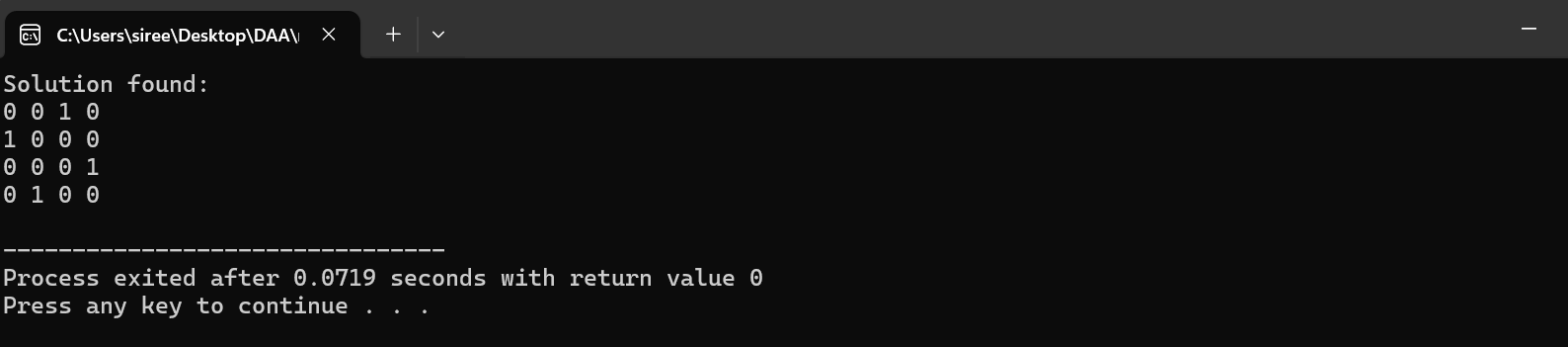
} else {

printf("No solution found.\n");

}

}

Output:



**33.Write a program to insert a number in the list.**

**Program:**

#include<stdio.h>

int main()

{

int arr[]={1,2,3};

int n=3;

int num=4;

arr[n]=num;

n++;

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

{

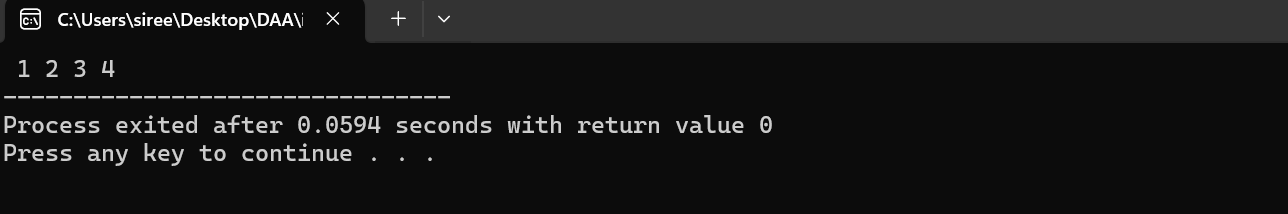
printf(" %d",arr[i]);

}

return 0;

}

**Output:**

****

**34.Write a program to perform sum of the subsets problem using backtracking.**

**Program:**

#include <stdio.h>

void subsetSum(int set[], int n, int sum, int i, int partialSum, int subset[]) {

if (partialSum == sum) {

printf("Subset: ");

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

if (subset[j]) printf("%d ", set[j]);

}

printf("\n");

}

if (i >= n || partialSum > sum) return;

subset[i] = 1;

subsetSum(set, n, sum, i + 1, partialSum + set[i], subset);

subset[i] = 0;

subsetSum(set, n, sum, i + 1, partialSum, subset);

}

int main() {

int n, sum;

printf("Enter number of elements: ");

scanf("%d", &n);

int set[n];

printf("Enter %d numbers: ", n);

for (int i = 0; i < n; i++) scanf("%d", &set[i]);

printf("Enter target sum: ");

scanf("%d", &sum);

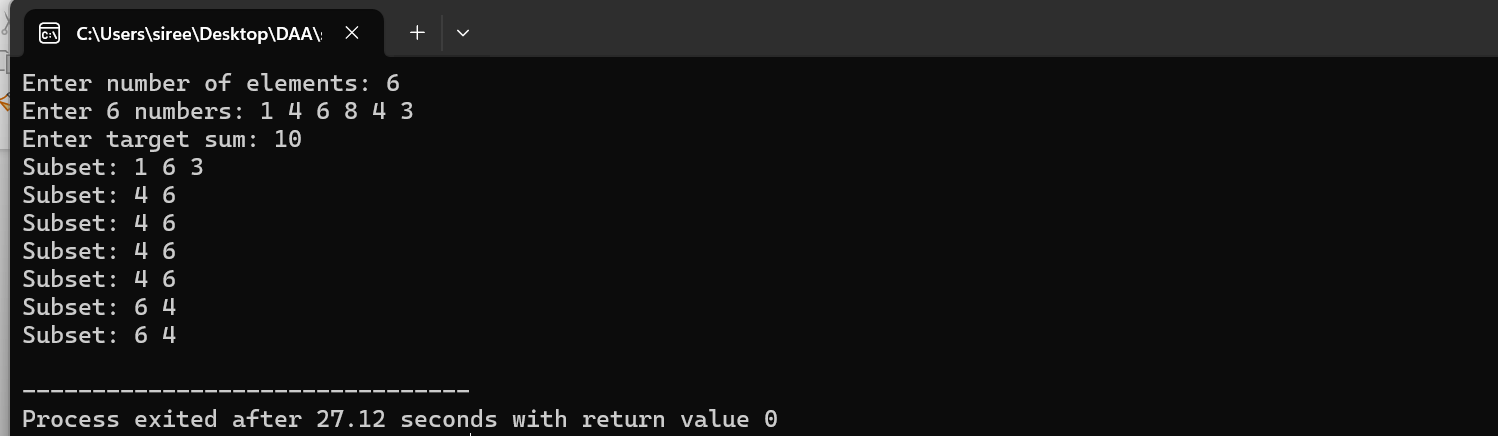
int subset[n] = {0};

subsetSum(set, n, sum, 0, 0, subset);

return 0;

}

**Output:**



**35.write a program to perform graph colouring using backtracking.**

**Program:**

#include <stdio.h>

#define MAX\_VERTICES 10

#define COLORS 3

int graph[MAX\_VERTICES][MAX\_VERTICES];

int color[MAX\_VERTICES];

void printColors(int n) {

printf("Colors: ");

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

printf("%d ", color[i]);

}

printf("\n");

}

int isValid(int v, int c, int n) {

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

if (graph[v][i] && color[i] == c) {

return 0;

}

}

return 1;

}

int graphColoring(int v, int n) {

if (v == n) {

printColors(n);

return 1;

}

for (int c = 1; c <= COLORS; c++) {

if (isValid(v, c, n)) {

color[v] = c;

if (graphColoring(v + 1, n)) {

return 1;

}

color[v] = 0;

}

}

return 0;

}

int main() {

int n;

printf("Enter number of vertices: ");

scanf("%d", &n);

printf("Enter adjacency matrix:\n");

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

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

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

}

}

if (!graphColoring(0, n)) {

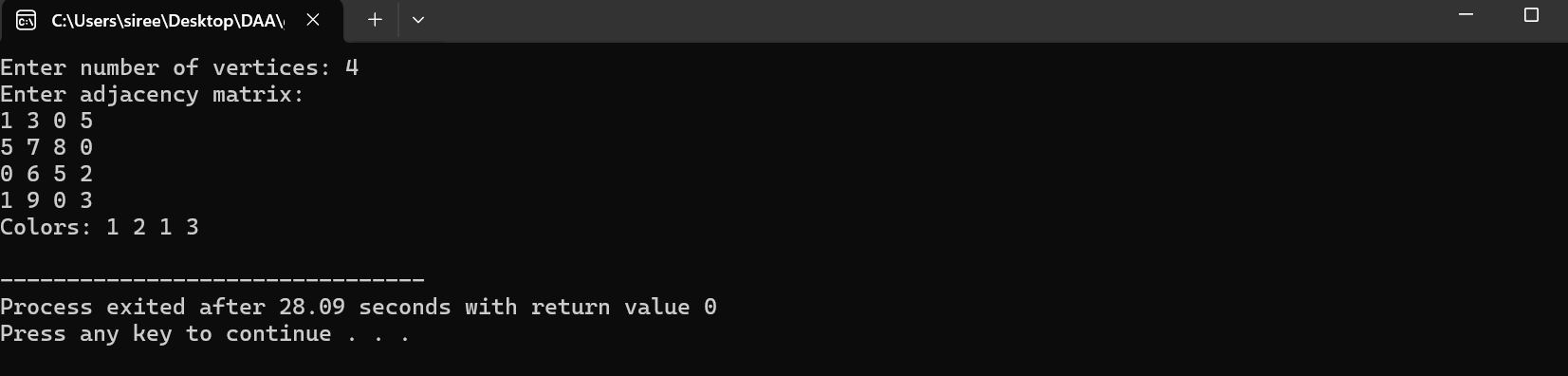
printf("No solution exists.\n");

}

return 0;

}

Output:



**36. Write a program to compute container loader problem.**

**Program:**

#include <stdio.h>

#define MAX\_CONTAINERS 100

void loadContainers(int weights[], int n, int maxWeight) {

int currentWeight = 0;

printf("Loaded containers: ");

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

if (currentWeight + weights[i] <= maxWeight) {

currentWeight += weights[i];

printf("%d ", weights[i]);

}

}

printf("\nTotal weight: %d\n", currentWeight);

}

int main() {

int weights[MAX\_CONTAINERS] = {10, 40, 20, 30, 50};

int n = 5;

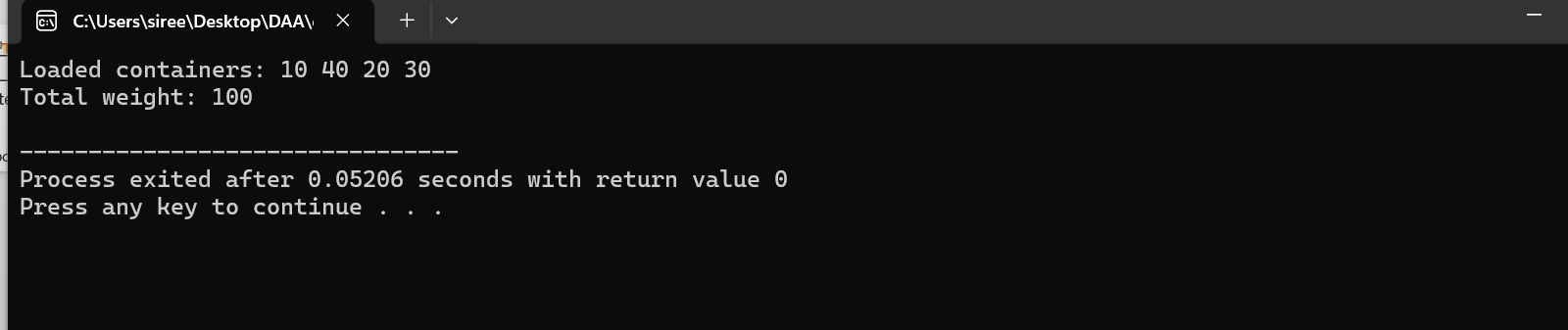
int maxWeight = 100;

loadContainers(weights, n, maxWeight);

return 0;

}

**Output:**

****

**37.write a program to generate a list of all factors for n value.**

**Program:**

#include <stdio.h>

int main() {

int n;

printf("Enter a number: ");

scanf("%d", &n);

printf("Factors of %d: ", n);

for (int i = 1; i <= n; i++) {

if (n % i == 0) printf("%d ", i);

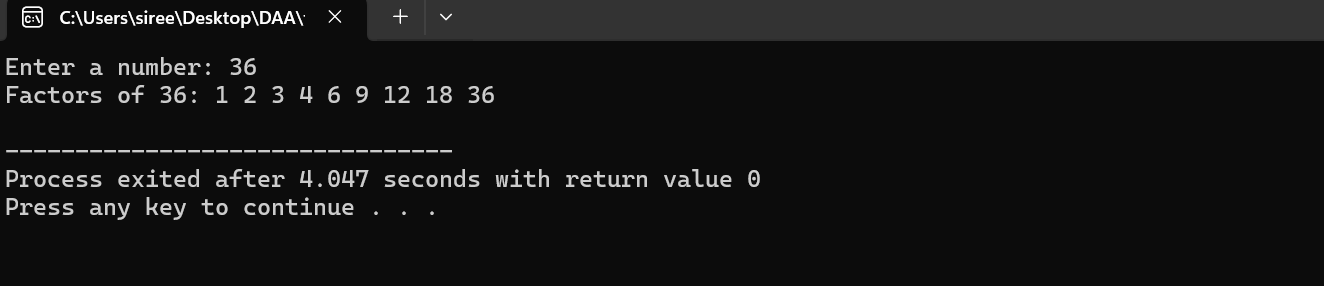
}

printf("\n");

return 0;

**}**

**Output:**



**38.write a program to perform assignment problem using branch and bound method.**

**Program:**

#include <stdio.h>

#include <limits.h>

#define N 4

int cost[N][N] = {

{10, 20, 30, 40},

{50, 60, 70, 80},

{90, 100, 110, 120},

{130, 140, 150, 160}

};

int minCost = INT\_MAX;

int assignment[N];

void branchAndBound(int row, int costSoFar, int taken[]) {

if (row == N) {

if (costSoFar < minCost) {

minCost = costSoFar;

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

assignment[i] = taken[i];

}

}

return;

}

for (int col = 0; col < N; col++) {

if (taken[col] == 0) {

taken[col] = 1;

branchAndBound(row + 1, costSoFar + cost[row][col], taken);

taken[col] = 0;

}

}

}

int main() {

int taken[N] = {0};

branchAndBound(0, 0, taken);

printf("Optimal Assignment:\n");

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

printf("Worker %d -> Task %d\n", i, assignment[i]);

}

printf("Minimum Cost: %d\n", minCost);

return 0;

}

Output:



**39.Write a program to perform linear search.**

**Program:**

#include <stdio.h>

int linearSearch(int arr[], int n, int target) {

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

if (arr[i] == target) {

return i;

}

}

return -1;

}

int main() {

int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};

int n = sizeof(arr) / sizeof(arr[0]);

int target;

printf("Enter target element: ");

scanf("%d", &target);

int result = linearSearch(arr, n, target);

if (result == -1) {

printf("Element not found.\n");

} else {

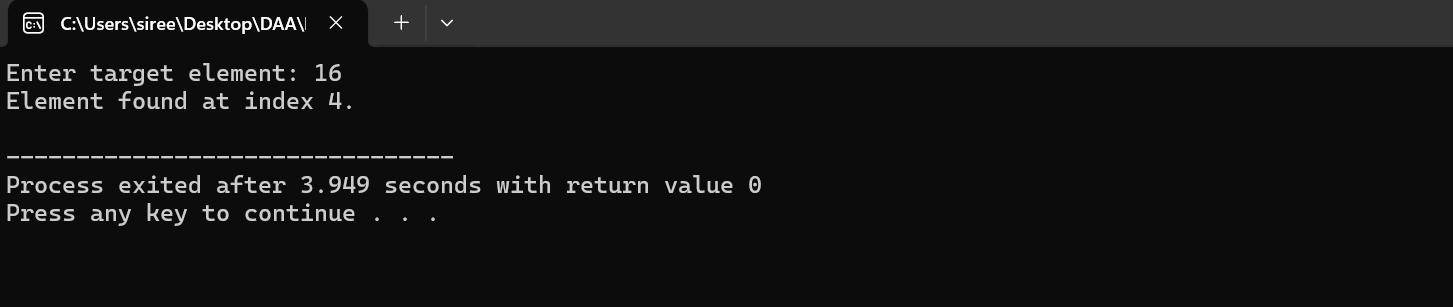
printf("Element found at index %d.\n", result);

}

return 0;

}

**Output:**

****

**40.Write a program to find out Hamiltonian circuit using backtracking.**

**Program:**

#include <stdio.h>

#define V 5

int graph[V][V] = {

{0, 1, 1, 1, 0},

{1, 0, 1, 1, 1},

{1, 1, 0, 0, 1},

{1, 1, 0, 0, 1},

{0, 1, 1, 1, 0}

};

int path[V];

void printPath() {

printf("Hamiltonian Circuit: ");

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

printf("%d ", path[i]);

}

printf("%d\n", path[0]);

}

int isValid(int v, int pos) {

if (graph[path[pos - 1]][v] == 0) {

return 0;

}

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

if (path[i] == v) {

return 0;

}

}

return 1;

}

int hamiltonianCycle(int pos) {

if (pos == V) {

if (graph[path[pos - 1]][path[0]] == 1) {

return 1;

}

return 0;

}

for (int i = 1; i < V; i++) {

if (isValid(i, pos)) {

path[pos] = i;

if (hamiltonianCycle(pos + 1)) {

return 1;

}

path[pos] = -1;

}

}

return 0;

}

int main() {

path[0] = 0;

if (!hamiltonianCycle(1)) {

printf("No Hamiltonian Circuit exists.\n");

} else {

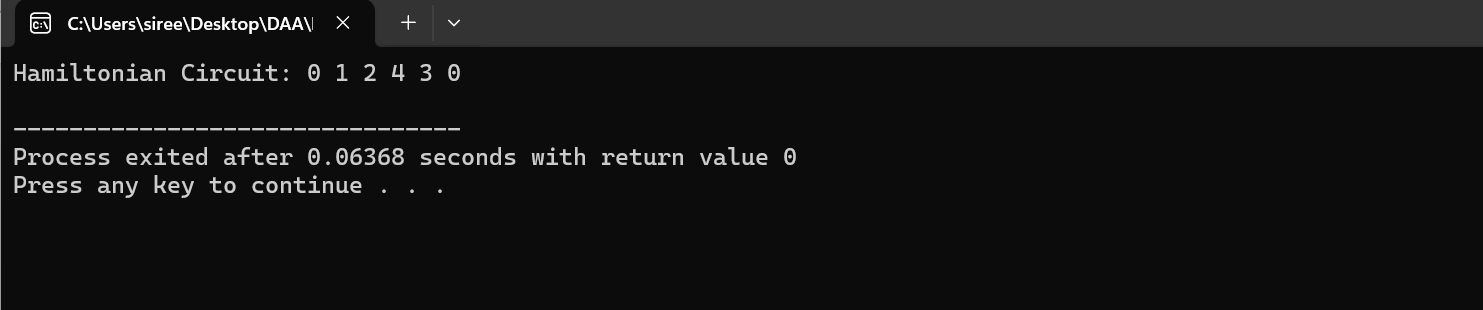
printPath();

}

return 0;

}

**Output:**

****