LAB PROGRAMS

DAY-1

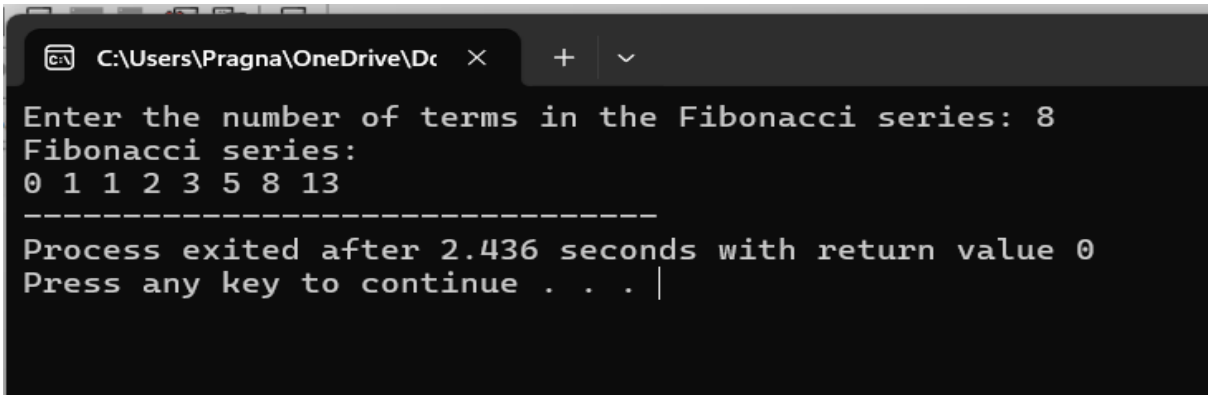
1.PRINT FIBONACCI SERIES USING RECURSION

A.

PROGRAM:

```
#include <stdio.h>
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
int main() {
    int n, i;

    printf("Enter the number of terms in the Fibonacci series: ");
    scanf("%d", &n);
    printf("Fibonacci series:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }

    return 0;
}
```

OUTPUT:



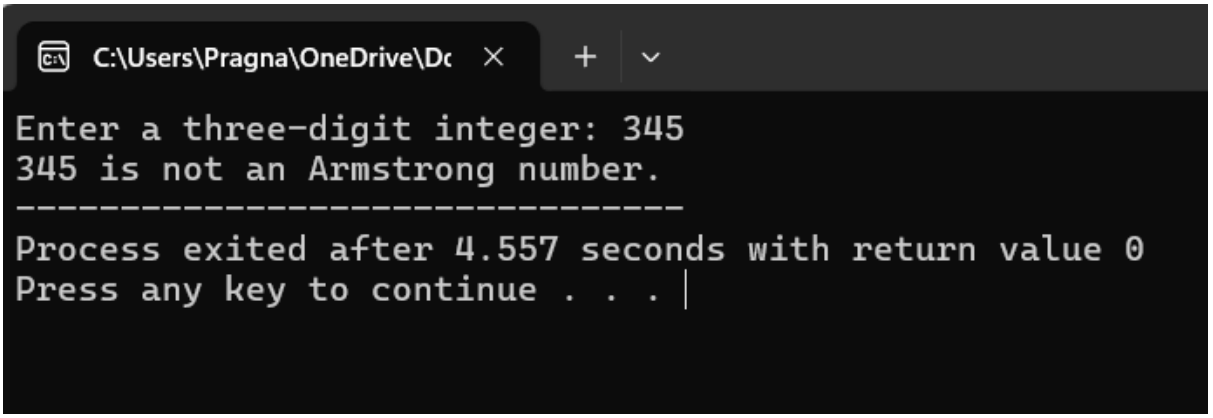
2.ARMSTRONG NUMBER OR NOT

A.

PROGRAM:

```
#include <stdio.h>
int main() {
    int num, originalNum, remainder, result = 0;
    printf("Enter a three-digit integer: ");
    scanf("%d", &num);
    original Num = num;

    while (originalNum != 0) {
        remainder = originalNum % 10;

       result += remainder * remainder * remainder;
       originalNum /= 10;
    }

    if (result == num)
        printf("%d is an Armstrong number.", num);
    else
        printf("%d is not an Armstrong number.", num);

    return 0;
}
```

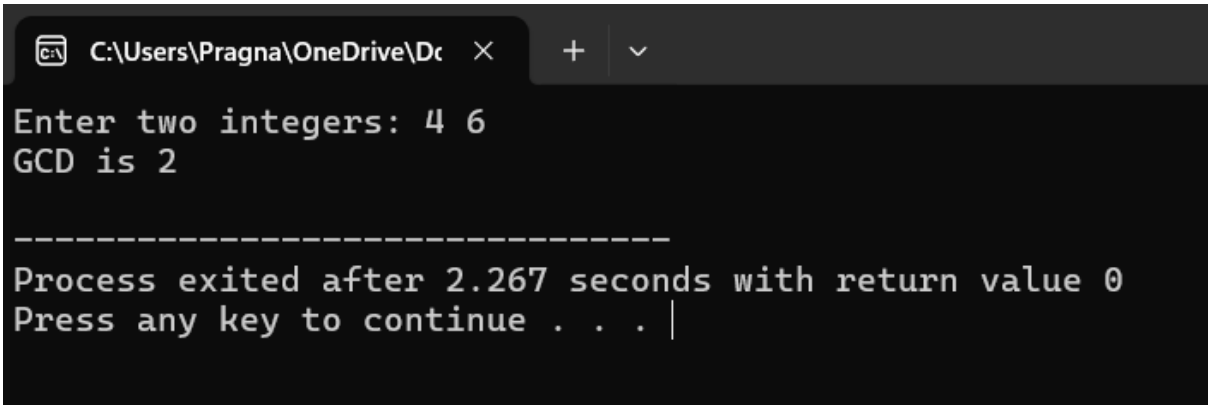
OUTPUT:



3.FIND GCD OF TWO NUMBERS
A
PROGRAM:

```
#include <stdio.h>

int main() {
    int a, b;

    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }

    printf("GCD is %d\n", a);
    return 0;
}
```

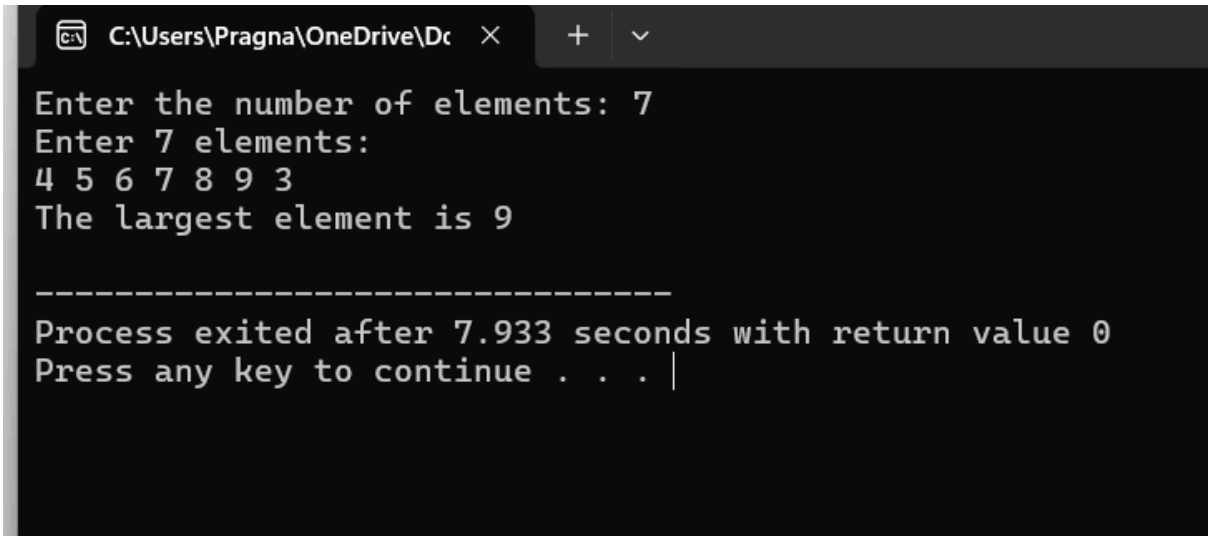
OUTPUT:



4.LARGEST ELEMENT IN A ARRAY
A.
PROGRAM:

```
#include <stdio.h>

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int largest = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > largest) {
            largest = arr[i];
        }
    }

    printf("The largest element is %d\n", largest);
    return 0;
}
```

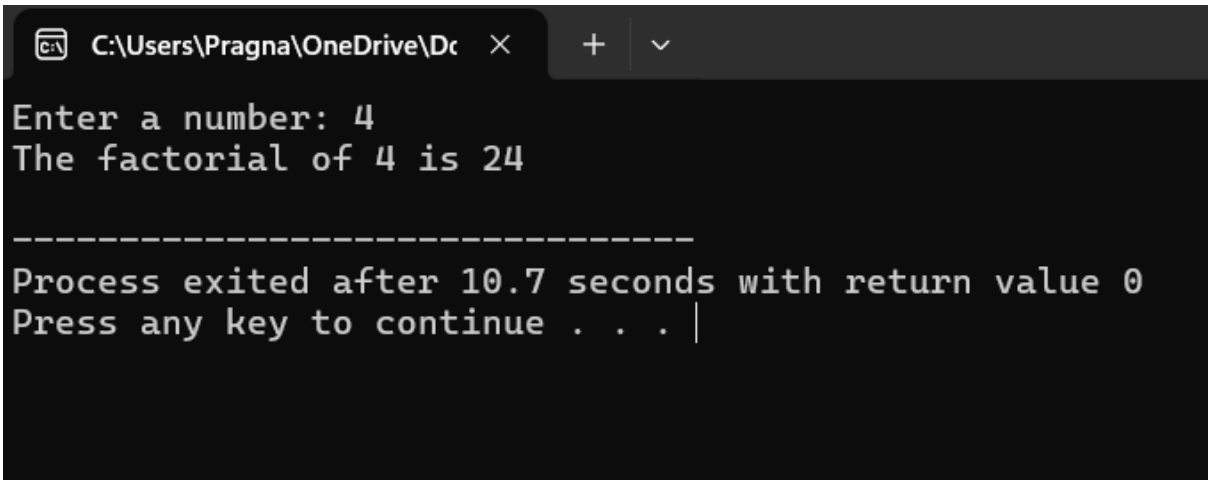
OUTPUT:



5.WRITE A PROGRAM TO FIND THE FACTORIAL OF A NUMBER
A
PROGRAM:

```
#include <stdio.h>

int main() {
    int num;
    unsigned long long factorial = 1;

    printf("Enter a number: ");
    scanf("%d", &num);

    if (num < 0) {
        printf("Factorial is not defined for negative numbers.\n");
    } else {
        for (int i = 1; i <= num; i++) {
            factorial *= i;
        }
        printf("The factorial of %d is %llu\n", num, factorial);
    }

    return 0;
}
```

OUTPUT:

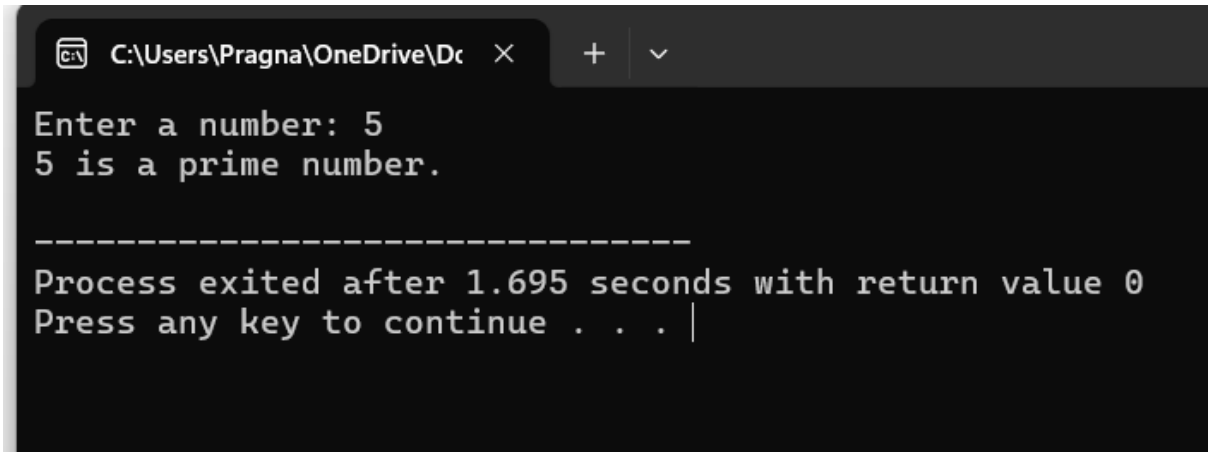


6.WRITE A PROGRAM TO CHECK A NUMBER IS A PRIME NUMBER OR NOT A

PROGRAM:

```
#include <stdio.h>

int main() {
    int num, isPrime = 1;

    printf("Enter a number: ");
    scanf("%d", &num);

    if (num <= 1) {
        isPrime = 0;
    } else {
        for (int i = 2; i * i <= num; i++) {
            if (num % i == 0) {
                isPrime = 0;
                break;
            }
        }
    }

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

    return 0;
}
```

OUTPUT:



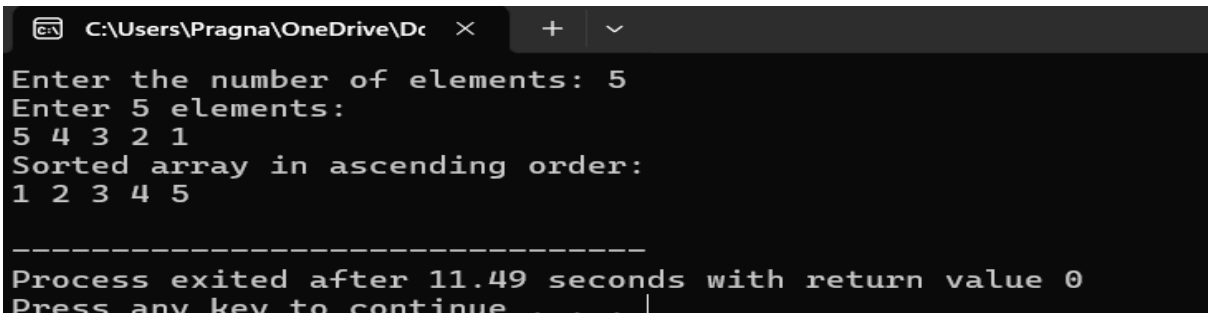
7.WRITE THE PROGRAM FOR SELECTION SORT

A.

PROGRAM:

```
#include <stdio.h>
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}
int main() {
    int n;
          printf("Enter the number of elements: ");
    scanf("%d", &n);
          int arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
selectionSort(arr, n);
          printf("Sorted array in ascending order:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
 return 0;
}
```

OUTPUT:



DAY-2

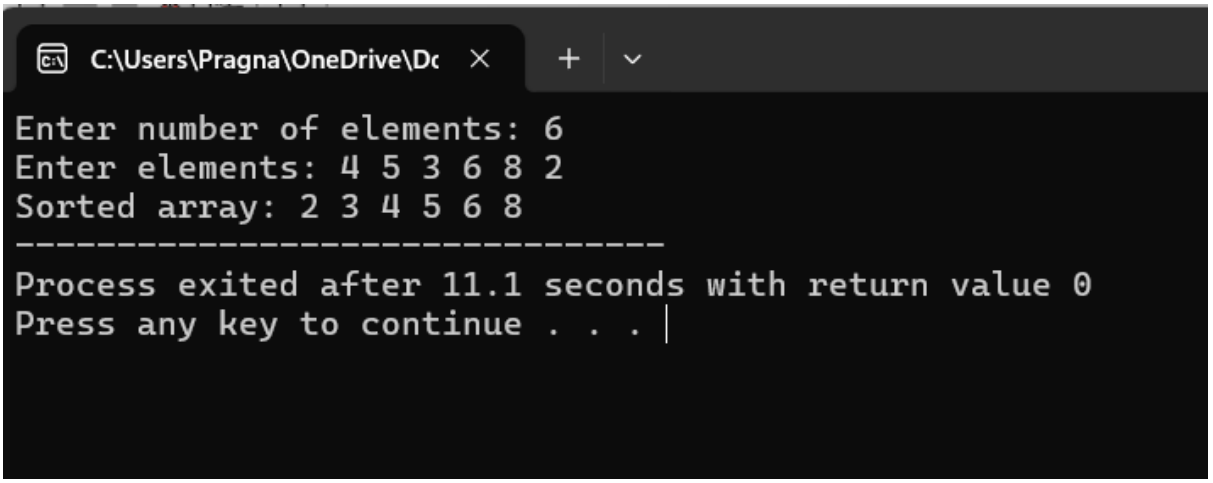
8.PRINT THE PROGRAM FOR BUBBLE SORT

A.

PROGRAM:

```
#include <stdio.h>
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
}
int main() {
    int n, arr[100];
          printf("Enter number of elements: ");
    scanf("%d", &n);
          printf("Enter elements: ");
    for (int i = 0; i < n; i++) scanf("%d", &arr[i]);
          bubbleSort(arr, n);
          printf("Sorted array: ");
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    return 0;
}
```

OUTPUT:

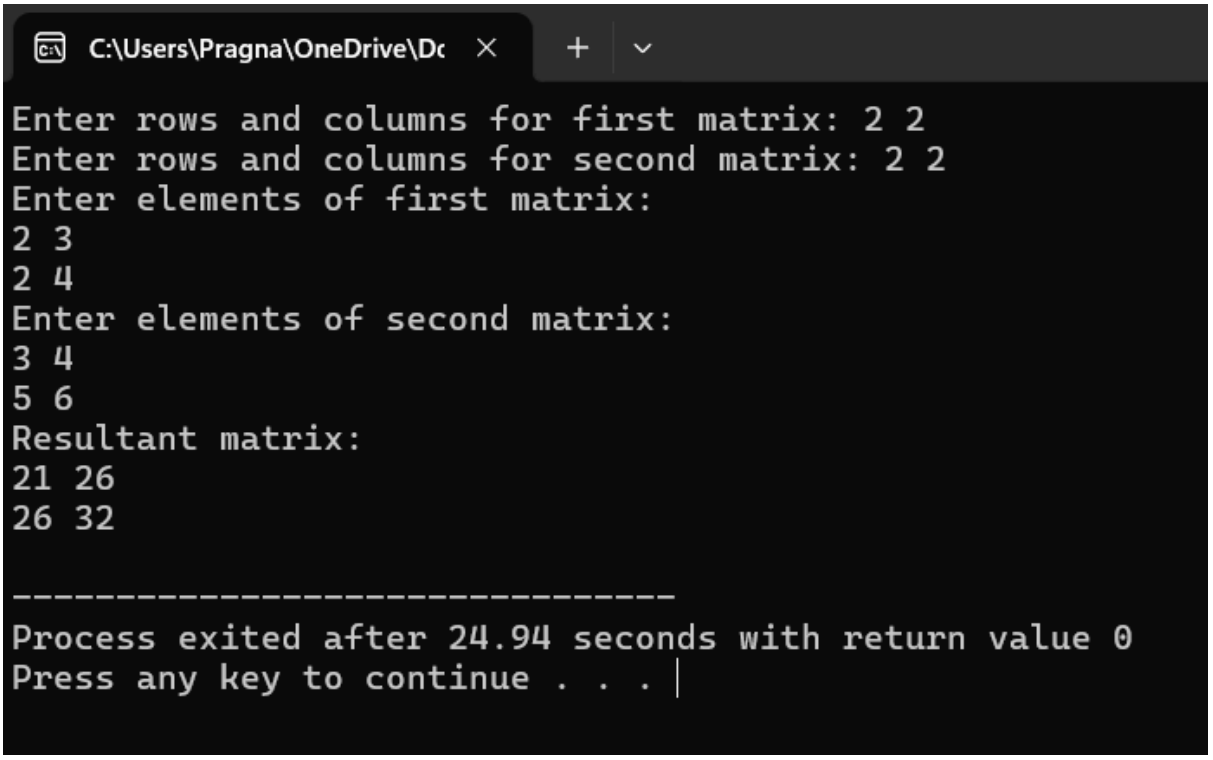


9.WRITE A PROGRAM MULTIPLY TWO MATRIX
A
PROGRAM:

```
#include <stdio.h>
int main() {
    int r1, c1, r2, c2;
    printf("Enter rows and columns for first matrix: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter rows and columns for second matrix: ");
    scanf("%d %d", &r2, &c2);
          if (c1 != r2) {
        printf("Matrix multiplication not possible.\n");
        return 1;
    }
          int a[r1][c1], b[r2][c2], result[r1][c2];
    printf("Enter elements of first matrix:\n");
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c1; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    printf("Enter elements of second matrix:\n");
    for (int i = 0; i < r2; i++) {
        for (int j = 0; j < c2; j++) {
            scanf("%d", &b[i][j]);
        }
    }
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            result[i][j] = 0;
        }
    }
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            for (int k = 0; k < c1; k++) {
                result[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    printf("Resultant matrix:\n");
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            printf("%d ", result[i][j]);
        }
```

```
        printf("\n");
    }
 return 0;
}
```

OUTPUT:

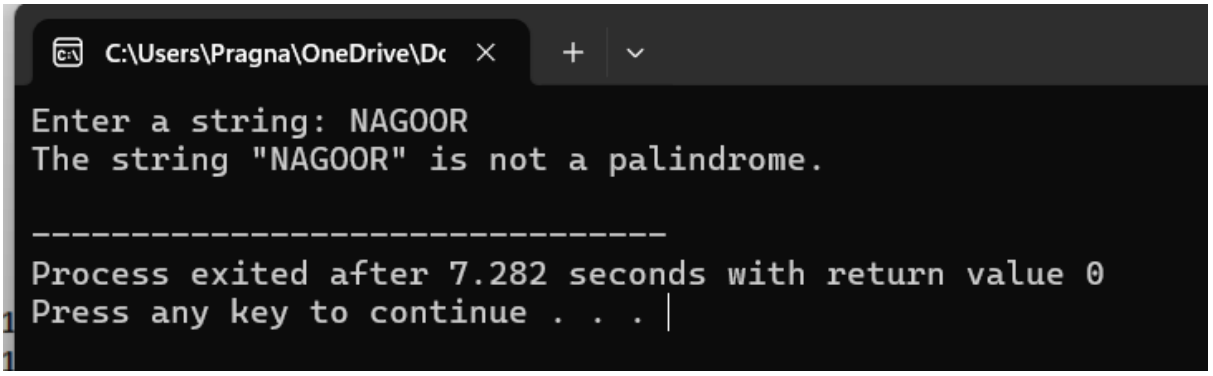


10.WRITE A PROGRAM THE GIVEN STRING IS PALINDROME OR NOT
A.
PROGRAM:

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[100], reversed[100];
    int length, isPalindrome = 1;

    printf("Enter a string: ");
    scanf("%s", str);

    length = strlen(str);

    for (int i = 0; i < length; i++) {
        if (str[i] != str[length - i - 1]) {
            isPalindrome = 0;
            break;
```

```
        }
    }

    if (isPalindrome)
        printf("The string \"%s\" is a palindrome.\n", str);
    else
        printf("The string \"%s\" is not a palindrome.\n", str);

    return 0;
}
```

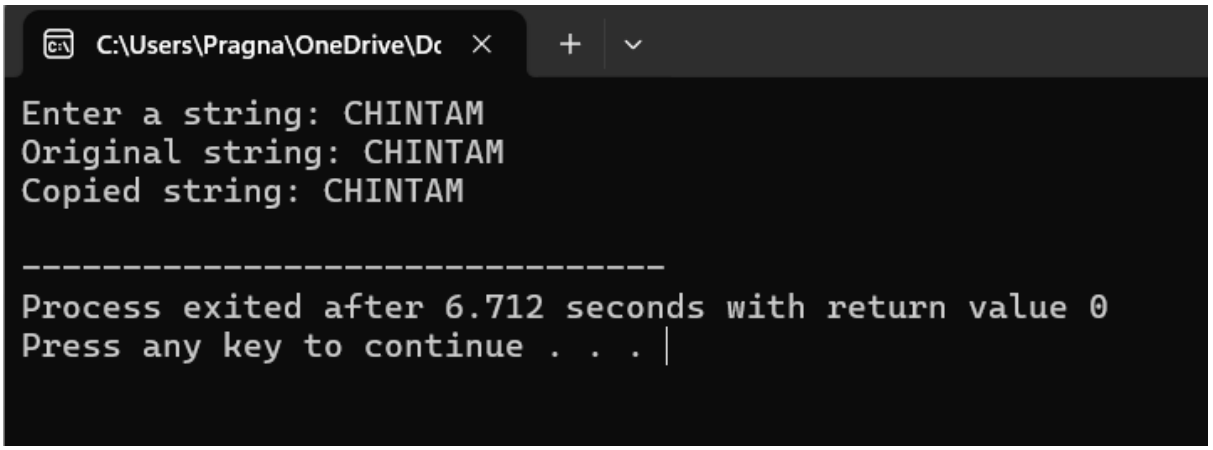
OUTPUT:



11.WRITE A PROGRAM TO COPY ONE STRING TO ANOTHER
A
PROGRAM:

```
#include <stdio.h>

int main() {
    char str1[100], str2[100];
    int i;

    printf("Enter a string: ");
    scanf("%s", str1);
    for (i = 0; str1[i] != '\0'; i++) {
        str2[i] = str1[i];
    }
    str2[i] = '\0';

    printf("Original string: %s\n", str1);
    printf("Copied string: %s\n", str2);

    return 0;
}
```

OUTPUT:



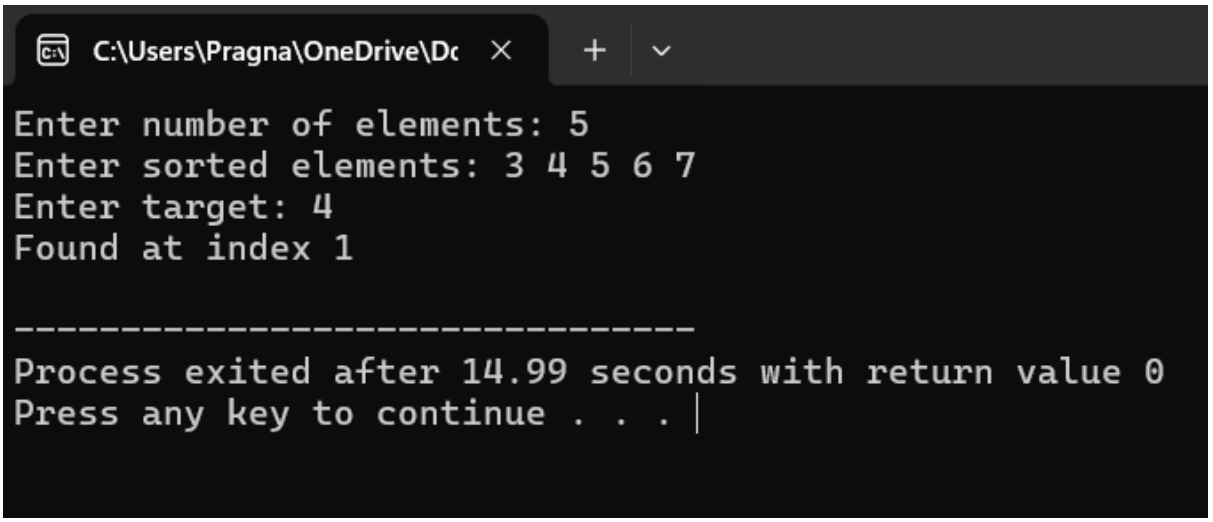
12.WRITE A PROGRAM TO PERFORM BINARY SEARCH

A.

PROGRAM:

```
#include <stdio.h>

int binarySearch(int arr[], int n, int target) {
    int left = 0, right = n - 1, mid;
    while (left <= right) {
        mid = left + (right - left) / 2;
        if (arr[mid] == target) return mid;
        else if (arr[mid] < target) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}

int main() {
    int n, arr[100], target;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter sorted elements: ");
    for (int i = 0; i < n; i++) scanf("%d", &arr[i]);
    printf("Enter target: ");
    scanf("%d", &target);

    int result = binarySearch(arr, n, target);
    result != -1 ? printf("Found at index %d\n", result) : printf("Not found\n");
    return 0;
}
```

OUTPUT:



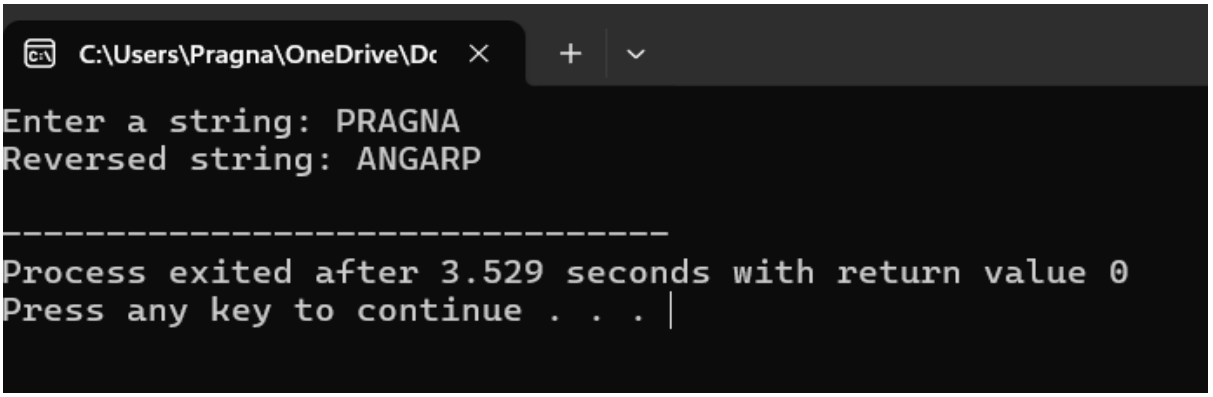
13.WRITE A PROGRAM TO REVERSE A STRING

A.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[100];

   printf("Enter a string: ");
    scanf("%s", str);

    int length = strlen(str);

    printf("Reversed string: ");
    for (int i = length - 1; i >= 0; i--) {
        printf("%c", str[i]);
    }
    printf("\n");

    return 0;
}
```

OUTPUT:



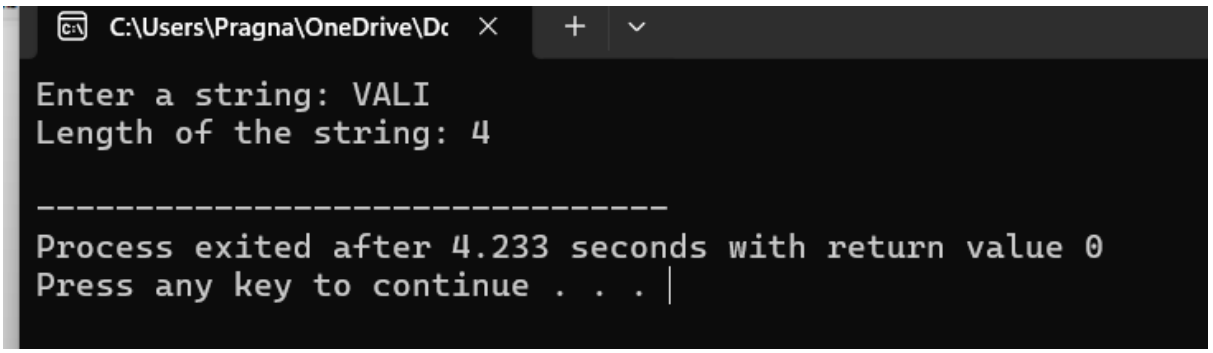
14.WRITE A PROGRAM TO FIND THE LENGTH OF A STRING

A.

PROGRAM:

```
#include <stdio.h>

int main() {
    char str[100];
    int length = 0;

    printf("Enter a string: ");
    scanf("%s", str);
    while (str[length] != '\0') {
        length++;
    }

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

    return 0;
}
```

OUTPUT:



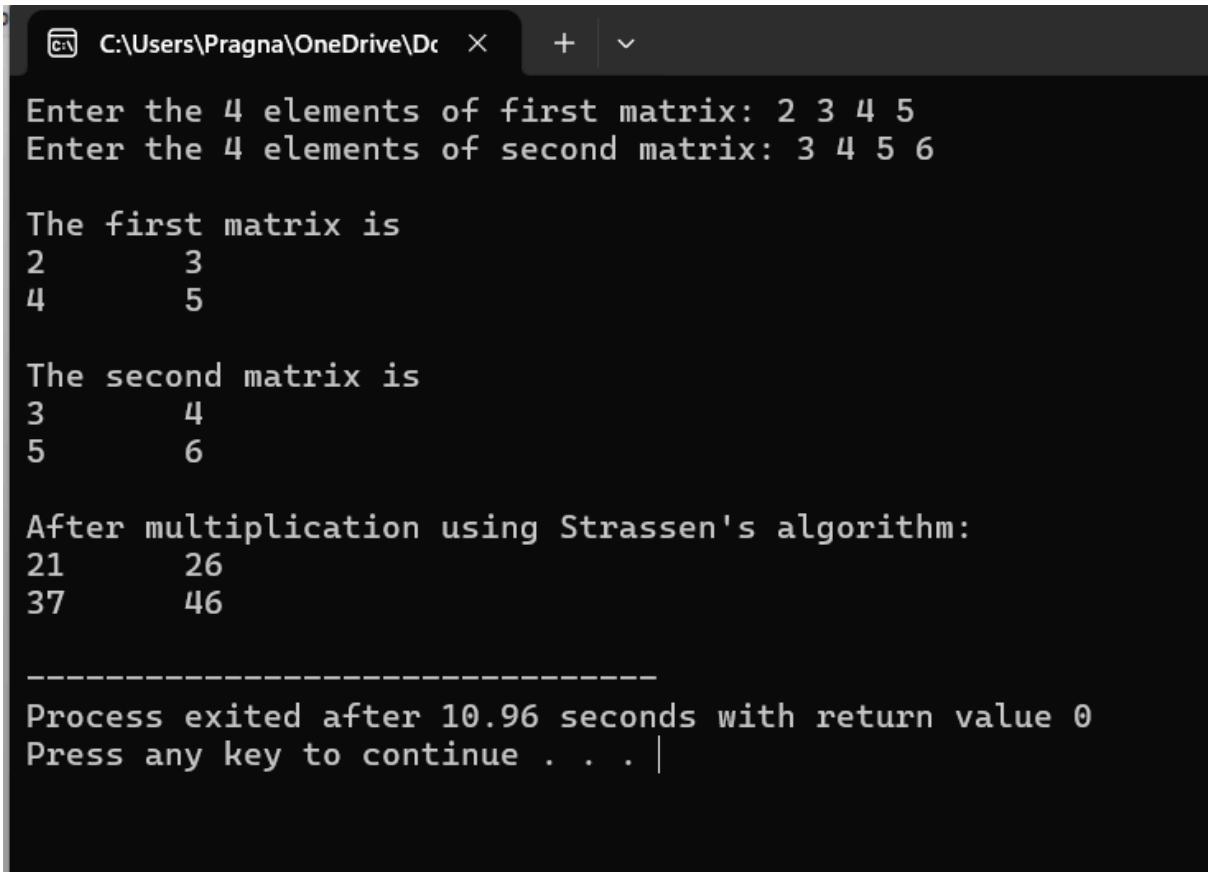
15.WRITE A PROGRAM TO PERFORM THE STARSSENS MATRIX A.

PROGRAM:

```
#include <stdio.h>
int main() {
    int a[2][2], b[2][2], c[2][2], i, j;
    int m1, m2, m3, m4, m5, m6, m7;
    printf("Enter the 4 elements of first matrix: ");
    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            scanf("%d", &a[i][j]);
    printf("Enter the 4 elements of second matrix: ");
    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++)
            scanf("%d", &b[i][j]);
    printf("\nThe first matrix is\n");
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++)
            printf("%d\t", a[i][j]);
        printf("\n");
    }
    printf("\nThe second matrix is\n");
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++)
            printf("%d\t", b[i][j]);
        printf("\n");
    }
    m1 = (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
    m2 = (a[1][0] + a[1][1]) * b[0][0];
    m3 = a[0][0] * (b[0][1] - b[1][1]);
    m4 = a[1][1] * (b[1][0] - b[0][0]);
    m5 = (a[0][0] + a[0][1]) * b[1][1];
    m6 = (a[1][0] - a[0][0]) * (b[0][0] + b[0][1]);
    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;
    printf("\nAfter multiplication using Strassen's algorithm:\n");
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++)
            printf("%d\t", c[i][j]);
        printf("\n");
    }
```

```
    return 0;
}
```

OUTPUT:



DAY-3

16.WRITE A PROGRAM TO PERFORM A MERGE SORT

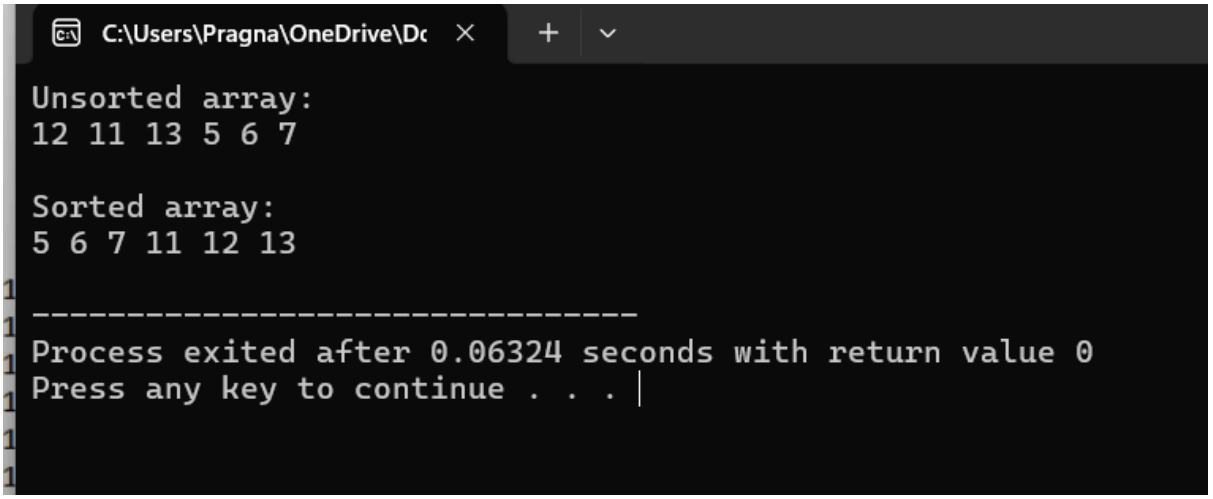
A.

PROGRAM:

```
#include <stdio.h>
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int L[n1], R[n2];
    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
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);
    }
}
```

```
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr) / sizeof(arr[0]);
          printf("Unsorted array: \n");
    printArray(arr, arr_size);
          mergeSort(arr, 0, arr_size - 1);
          printf("\nSorted array: \n");
    printArray(arr, arr_size);
    return 0;
}
```

OUTPUT:



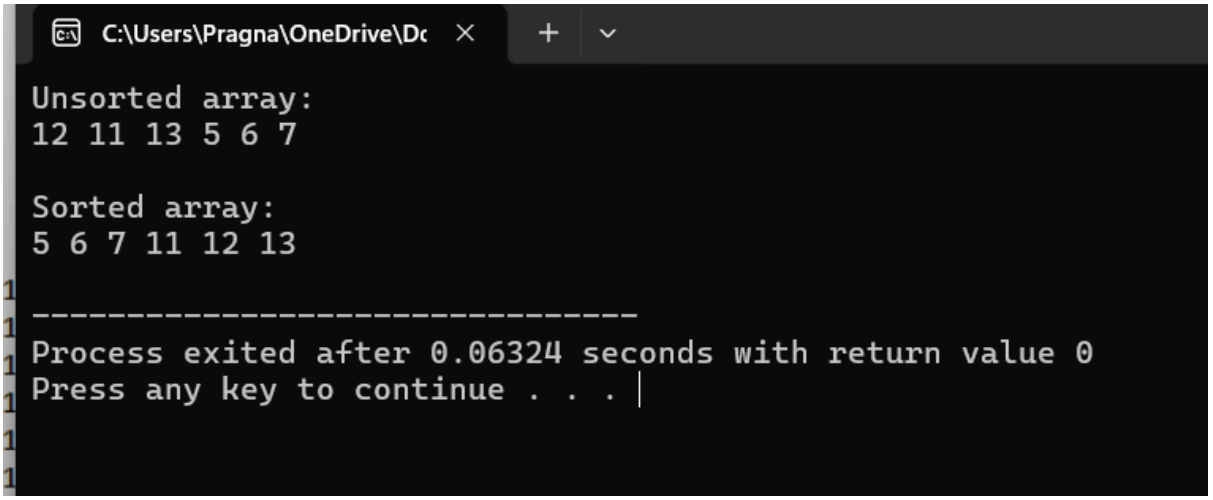
17.USING DIVIDE AND CONQUER METHOD TO FIND MIN AND MAX IN A LIST A.

PROGRAM:

```
#include <stdio.h>

void findMaxMin(int arr[], int left, int right, int *max, int *min) {
    if (left == right) {
        *max = arr[left];
        *min = arr[left];
    } else if (left + 1 == right) {
        if (arr[left] > arr[right]) {
            *max = arr[left];
            *min = arr[right];
        } else {
```

```
            *max = arr[right];
            *min = arr[left];
        }
    } else {
        int mid = (left + right) / 2;
        int leftMax, leftMin, rightMax, rightMin;
        findMaxMin(arr, left, mid, &leftMax, &leftMin);
        findMaxMin(arr, mid + 1, right, &rightMax, &rightMin);
        *max = (leftMax > rightMax) ? leftMax : rightMax;
        *min = (leftMin < rightMin) ? leftMin : rightMin;
    }
}

int main() {
    int arr[] = {12, 34, 8, 9, 56, 1, 45, 78};
    int n = sizeof(arr) / sizeof(arr[0]);
    int max, min;

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

    printf("Maximum: %d\n", max);
    printf("Minimum: %d\n", min);

    return 0;
}
```

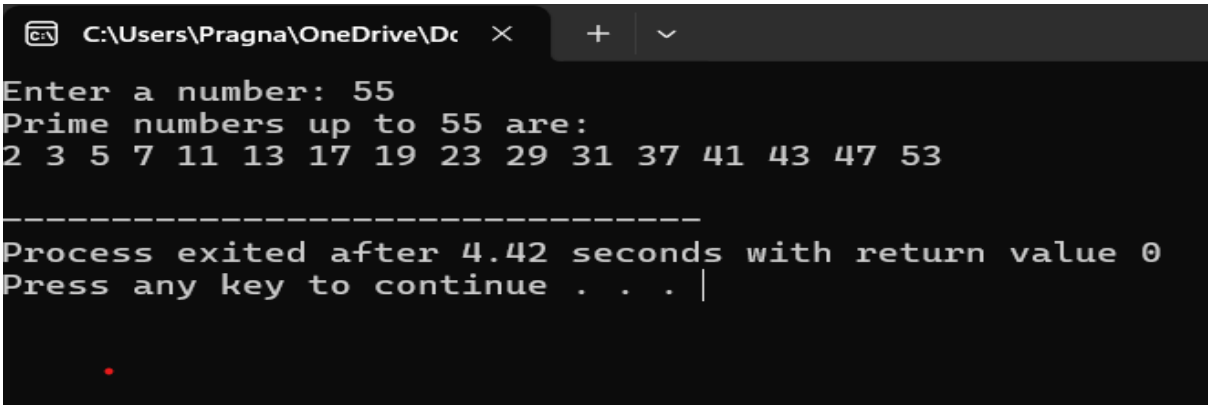
OUTPUT:



18.WRITE A PROGRAM TO GENERATE ALL PRIME NUMBER
A
PROGRAM:

```
#include <stdio.h>
int isPrime(int num) {
    if (num <= 1) {
        return 0;
    }
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            return 0;
        }
    }
    return 1;
}
void generatePrimes(int n) {
    printf("Prime numbers up to %d are:\n", n);
    for (int i = 2; i <= n; i++) {
        if (isPrime(i)) {
            printf("%d ", i);
        }
    }
    printf("\n");
}
int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);

    generatePrimes(n);

    return 0;
}
```

OUTPUT:

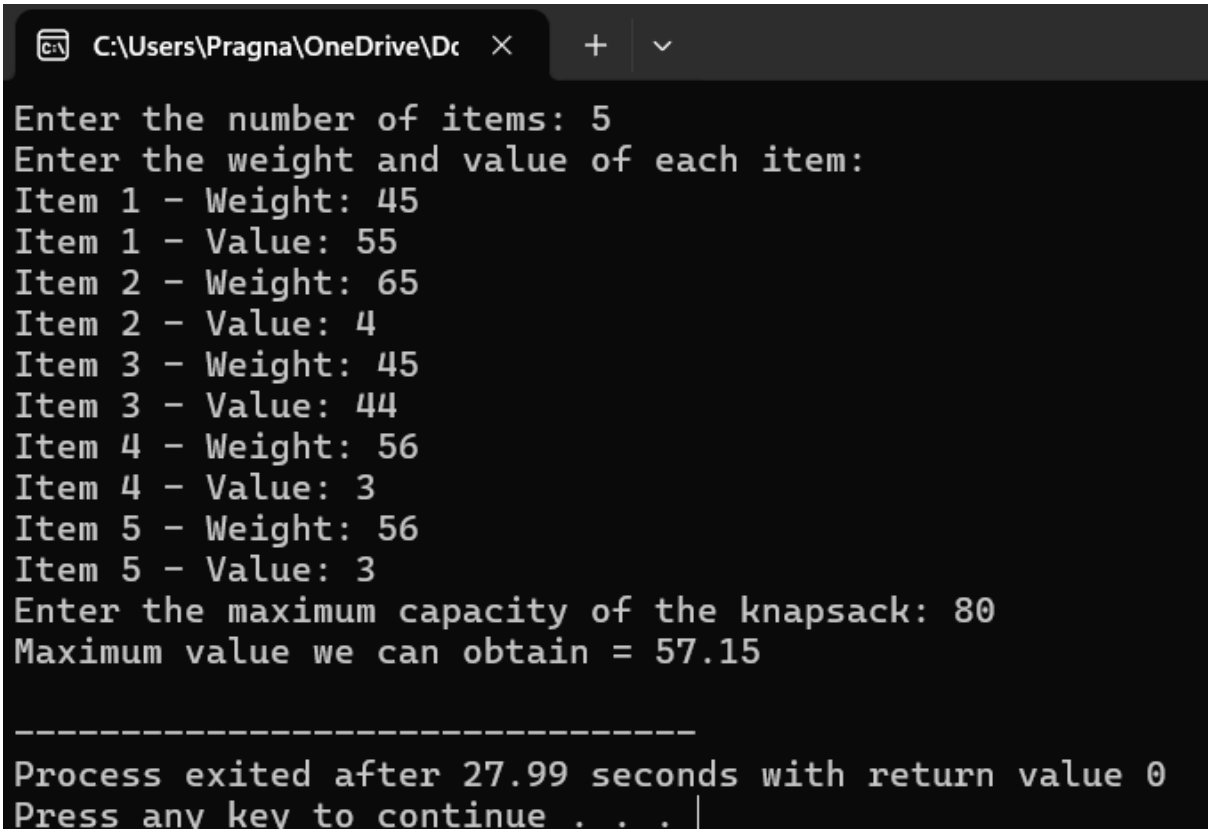


19.WRITE A PROGRAM USING KNAPSACK USING GREEDY
A.
PROGRAM:

```
#include <stdio.h>
typedef struct {
    int weight;
    int value;
    float ratio;
} Item;
void calculateRatio(Item items[], int n) {
    for (int i = 0; i < n; i++) {
        items[i].ratio = (float)items[i].value / items[i].weight;
    }
}
int compare(const void *a, const void *b) {
    Item *item1 = (Item *)a;
    Item *item2 = (Item *)b;
    if (item1->ratio > item2->ratio) return -1;
    if (item1->ratio < item2->ratio) return 1;
    return 0;
}
float fractionalKnapsack(int W, Item items[], int n) {

    float totalValue = 0.0;
    for (int i = 0; i < n; i++) {
        if (items[i].weight <= W) {
            W -= items[i].weight;
            totalValue += items[i].value;
        }
        else {
            totalValue += items[i].value * ((float)W / items[i].weight);
            break;
        }
    }
    return totalValue;
}
int main() {
    int n, W;
    printf("Enter the number of items: ");
    scanf("%d", &n);
    Item items[n];
    printf("Enter the weight and value of each item:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d - Weight: ", i+1);
```

```
        scanf("%d", &items[i].weight);
        printf("Item %d - Value: ", i+1);
        scanf("%d", &items[i].value);
    }
    printf("Enter the maximum capacity of the knapsack: ");
    scanf("%d", &W);
    calculateRatio(items, n);
    float maxValue = fractionalKnapsack(W, items, n);
    printf("Maximum value we can obtain = %.2f\n", maxValue);
    return 0;
}
```

OUTPUT:



20.WRITE THE PROGRAM TO MST USING GREEDY METHOD

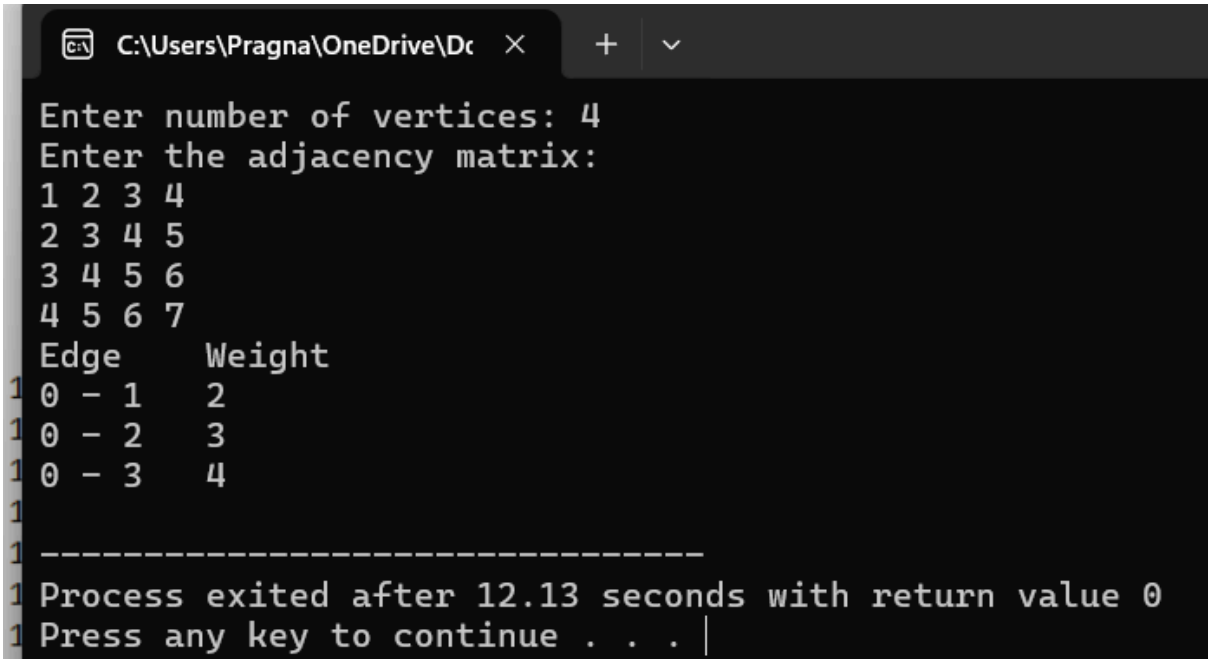
A.

PROGRAM:

```
#include <stdio.h>
#include <limits.h>
#define MAX 100
#define INF INT_MAX
int graph[MAX][MAX], parent[MAX], key[MAX], visited[MAX];
int n;
void primMST() {
    for (int i = 0; i < n; i++) {
```

```
        key[i] = INF;
        visited[i] = 0;
    }
key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < n - 1; count++) {
        int u = -1;
        for (int i = 0; i < n; i++) {
            if (!visited[i] && (u == -1 || key[i] < key[u]))
                u = i;
        }
 visited[u] = 1;

        for (int v = 0; v < n; v++) {
            if (graph[u][v] && !visited[v] && graph[u][v] < key[v]) {
                key[v] = graph[u][v];
                parent[v] = u;
            }
        }
    }

    printf("Edge \tWeight\n");
    for (int i = 1; i < n; i++) {
        printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
    }
}

int main() {
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    primMST();

    return 0;
}
```

OUTPUT:



21.DYNAMIC PROGRAMMING CONCEPT TO FIND THE OPTIMAL BINARY SEARCH TREE

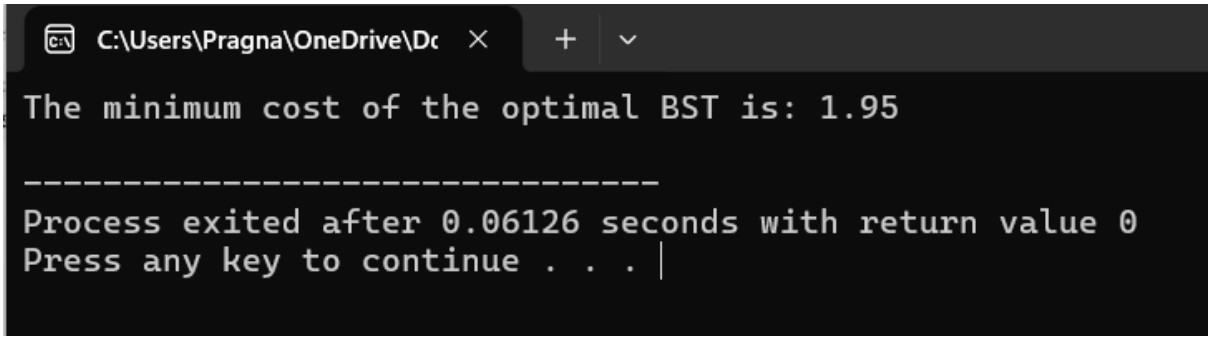
A.

PROGRAM:

```
#include <stdio.h>
#include <limits.h>
#define MAX 100
void optimalBST(int n, float p[], float q[], float *cost) {
    float e[MAX][MAX], w[MAX][MAX];
    int root[MAX][MAX];
    for (int i = 0; i <= n; i++) {
        e[i][i - 1] = q[i];
        w[i][i - 1] = q[i];
    }
    for (int l = 1; l <= n; l++) {
        for (int i = 1; i <= n - l + 1; i++) {
            int j = i + l - 1;
            e[i][j] = INT_MAX;
            w[i][j] = w[i][j - 1] + p[j] + q[j];
            for (int r = i; r <= j; r++) {
                float temp = e[i][r - 1] + e[r + 1][j] + w[i][j];
                if (temp < e[i][j]) {
                    e[i][j] = temp;
                    root[i][j] = r;
                }
            }
        }
```

```
 }
*cost = e[1][n];
    printf("The minimum cost of the optimal BST is: %.2f\n", *cost);
}

int main() {
    int n = 4;
    float p[] = {0.0, 0.15, 0.10, 0.05, 0.10};
    float q[] = {0.05, 0.10, 0.05, 0.05, 0.10};
    float cost;
optimalBST(n, p, q, &cost);
return 0;
}
```

OUTPUT:



22.USING DYNAMIC PROGRAMMING TECHNIQUES TO FIND BINOMIAL COEFFICIENT OF A GIVEN NUMBER

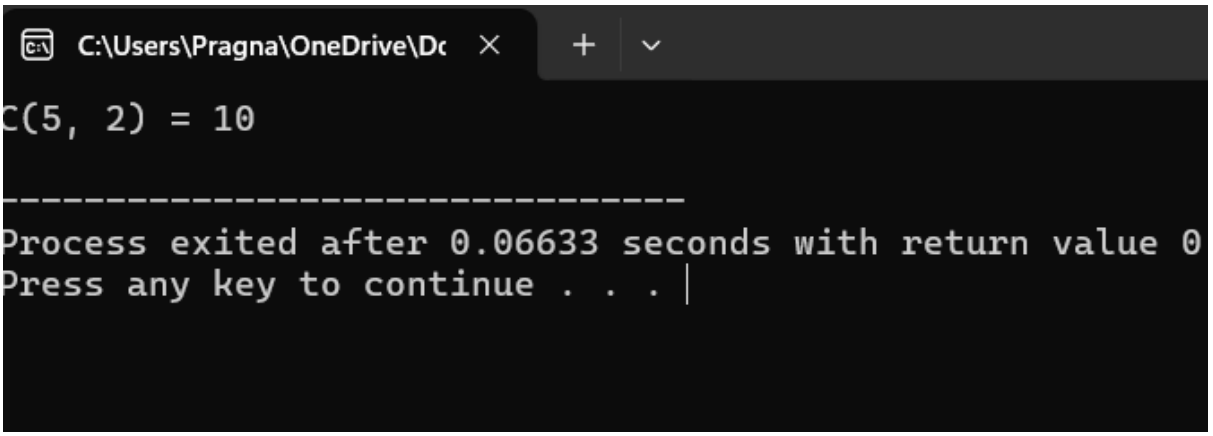
A

PROGRAM:

```
#include <stdio.h>
int binomialCoefficient(int n, int k) {
    int C[n + 1][k + 1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= (i < k ? i : k); j++) {
            if (j == 0 || j == i)
                C[i][j] = 1; // Base cases
            else
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
        }
    }

    return C[n][k];
}
```

```
int main() {
    int n = 5, k = 2;

    printf("C(%d, %d) = %d\n", n, k, binomialCoefficient(n, k));

    return 0;
}
```

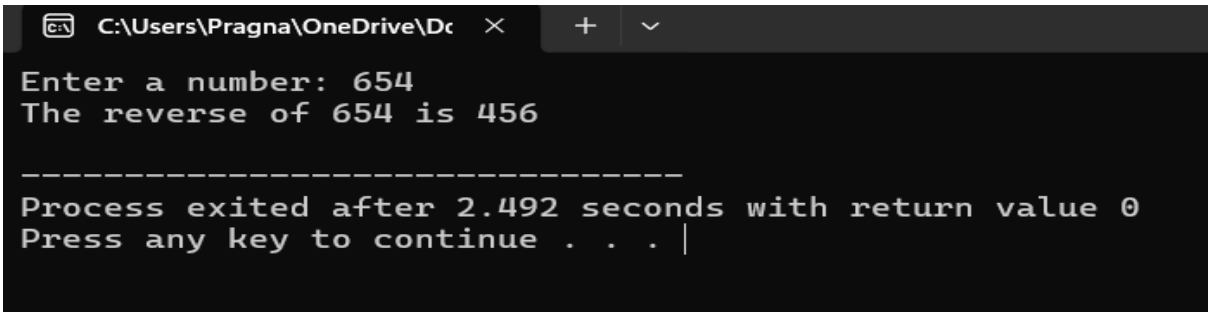
OUTPUT:



23.WRITE A PROGRAM TO REVERSE A NUMBER
A
PROGRAM:

```
#include <stdio.h>
int reverseNumber(int num) {
    int reverse = 0;

    while (num != 0) {
        int digit = num % 10;
        reverse = reverse * 10 + digit;
        num /= 10;
    }
    return reverse;
}
int main() {
    int number;
    printf("Enter a number: ");
    scanf("%d", &number);
    int reversed = reverseNumber(number);
    printf("The reverse of %d is %d\n", number, reversed);
          return 0;
}
```

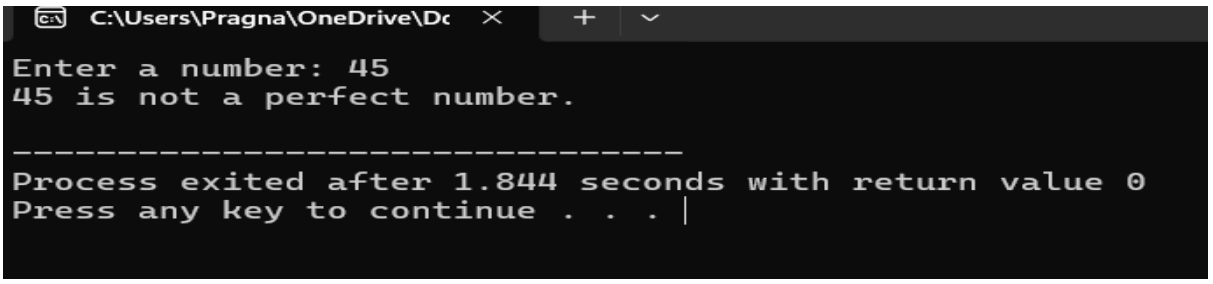
OUTPUT:



24.WRITE A PROGRAM TO FIND THE PERFECT NUMBER
A
PROGRAM:

```
#include <stdio.h>
int isPerfectNumber(int num) {
    int sum = 0;
    for (int i = 1; i <= num / 2; i++) {
        if (num % i == 0) {
            sum += i;
        }
    }
    return sum == num;
}
int main() {
    int number;
    printf("Enter a number: ");
    scanf("%d", &number);
    if (isPerfectNumber(number)) {
        printf("%d is a perfect number.\n", number);
    } else {
        printf("%d is not a perfect number.\n", number);
    }

    return 0;
}
```

OUTPUT:



25.WRITE A PROGRAM TO PERFORM TRAVELLING SALESMAN PROBLEM USING DYNAMIC PROGRAMMING

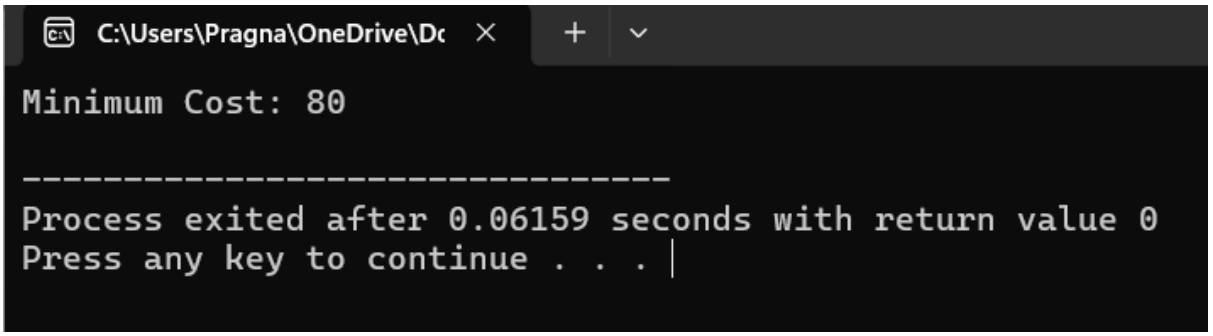
A

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define INF INT_MAX
int min(int a, int b) {
    return (a < b) ? a : b;
}
int tsp(int** distance, int n) {
    int size = 1 << n;
    int** dp = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        dp[i] = (int*)malloc(size * sizeof(int));
        for (int j = 0; j < size; j++) {
            dp[i][j] = INF;
        }
    }
    for (int i = 0; i < n; i++) {
        dp[i][1 << i] = distance[i][0];
    }
    for (int visited = 1; visited < size; visited++) {
        for (int current = 0; current < n; current++) {
            if (!(visited & (1 << current))) {
                continue;
            }
            for (int next = 0; next < n; next++) {
                if (visited & (1 << next)) {
                    continue;
                }
                int newVisited = visited | (1 << next);
                    dp[next][newVisited] = min(dp[next][newVisited], dp[current][visited] +
distance[current][next]);
            }
        }
    }
    int minCost = INF;
    for (int i = 1; i < n; i++) {
        minCost = min(minCost, dp[i][size - 1] + distance[i][0]);
    }
    for (int i = 0; i < n; i++) {
        free(dp[i]);
```

```
    }
    free(dp);
        return minCost;
}
int main() {
    int n = 4;
    int** distance = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        distance[i] = (int*)malloc(n * sizeof(int));
    }
    int distMatrix[4][4] = {
        {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}ER
    };
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            distance[i][j] = distMatrix[i][j];
        }
    }
    int minCost = tsp(distance, n);
    printf("Minimum Cost: %d\n", minCost);
    for (int i = 0; i < n; i++) {
        free(distance[i]);
    }
    free(distance);

    return 0;
}
```

OUTPUT:



DAY-4

26.WRITE A PROGRAM FOR GIVEN PATTERN

IF N=4

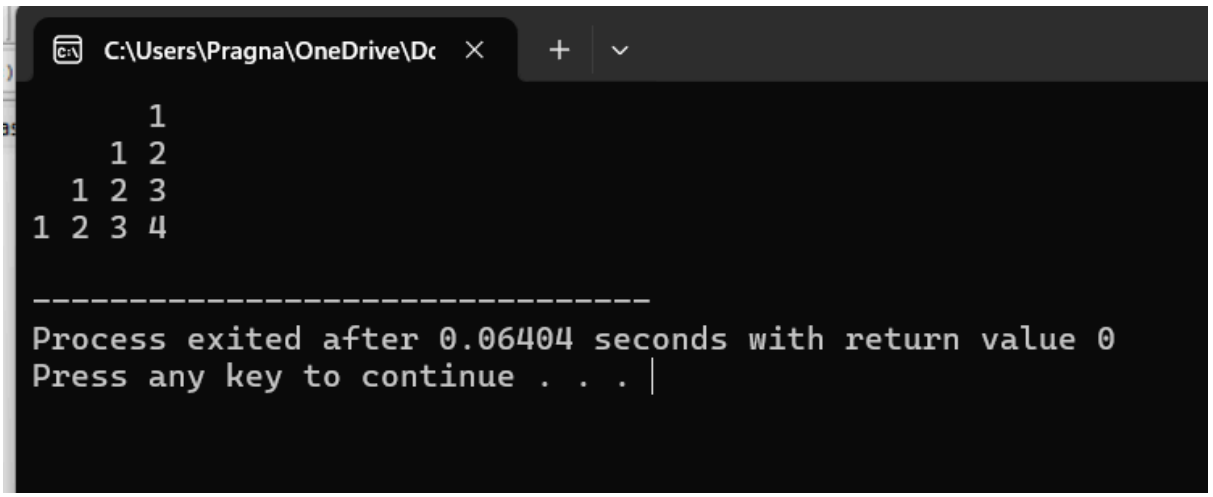
```
   1
  1  2
 1  2  3
1  2  3  4
```

A.

PROGRAM:

```
#include <stdio.h>
void print_pattern(int n) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n - i; j++) {
            printf("  ");
        }
        for (int j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        printf("\n");
    }
}
int main() {
 int n = 4;
    print_pattern(n);
    return 0;
}
```

OUTPUT:



27.WRITE A PROGRAM TO PERFORM FLOYD'S ALGORITHM

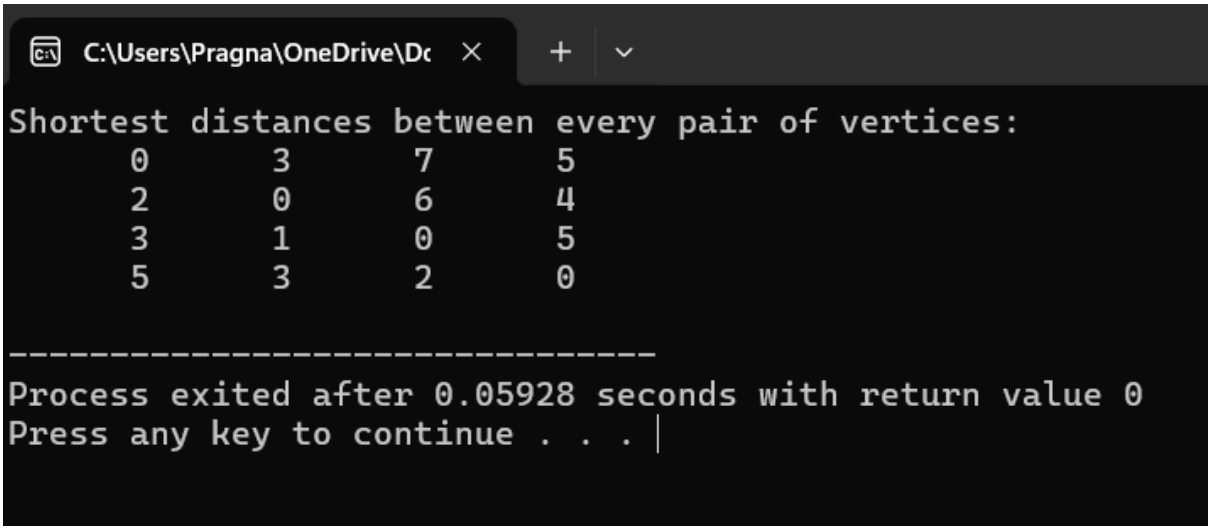
A.

PROGRAM:

```
#include <stdio.h>
#define INF 99999
#define MAX 10
void floydWarshall(int graph[MAX][MAX], int n) {
   int dist[MAX][MAX];
   int i, j, k;
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         dist[i][j] = graph[i][j];
      }
   }
   for (k = 0; k < n; k++) {
      for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++) {
            if (dist[i][k] + dist[k][j] < dist[i][j]) {
               dist[i][j] = dist[i][k] + dist[k][j];
```

```
            }
        }
    }
}
    printf("Shortest distances between every pair of vertices:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (dist[i][j] == INF) {
                printf("%7s", "INF");
            } else {
                printf("%7d", dist[i][j]);
            }
        }
        printf("\n");
    }
}
int main() {
    int graph[MAX][MAX] = {
        {0, 3, INF, 5},
        {2, 0, INF, 4},
        {INF, 1, 0, INF},
        {INF, INF, 2, 0}
    };
    int n = 4;
 floydWarshall(graph, n);
return 0;
}
```

OUTPUT:



28.WRITE A PROGRAM TO PERFORM PASCAL TRIANGLE

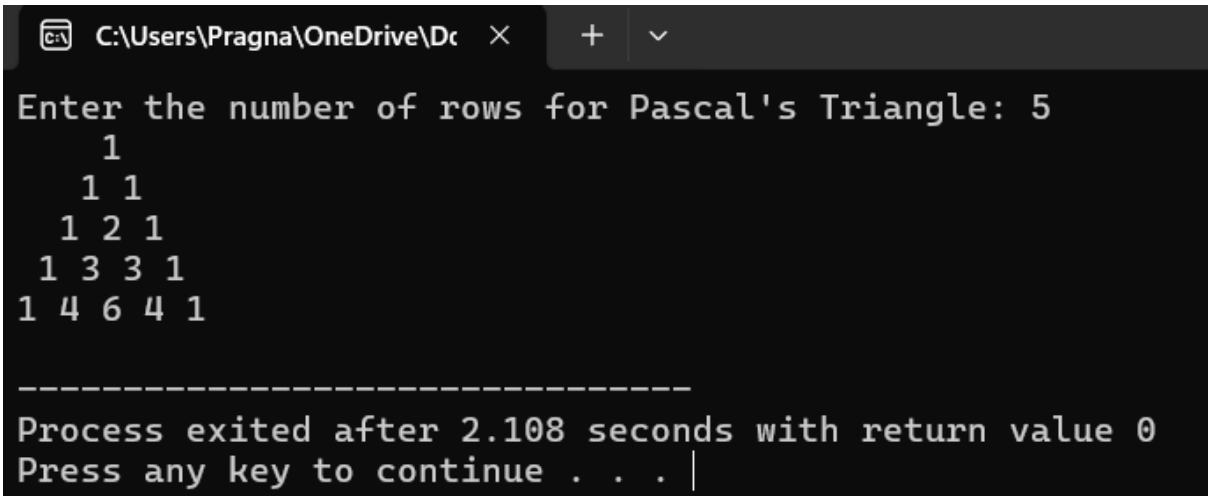
A

PROGRAM:

```
#include <stdio.h>

int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    }
    return n * factorial(n - 1);
}

void printPascalsTriangle(int rows) {
    for (int i = 0; i < rows; i++) {
        for (int space = 0; space < rows - i - 1; space++) {
            printf(" ");
        }
        for (int j = 0; j <= i; j++) {
            int value = factorial(i) / (factorial(j) * factorial(i - j));
            printf("%d ", value);
```

```
        }
        printf("\n");
    }
}

int main() {
    int rows;
    printf("Enter the number of rows for Pascal's Triangle: ");
    scanf("%d", &rows);
    printPascalsTriangle(rows);

    return 0;
}
```

OUTPUT:



29.WRITE A PROGRAM TO FIND OPTIMAL COST BY USING APPROPRIATE ALGORITHM

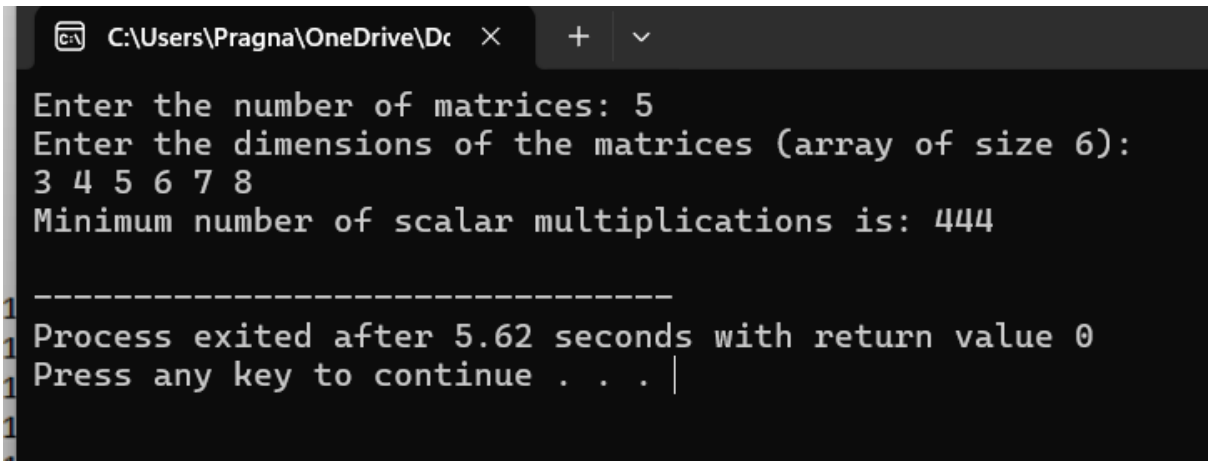
A

PROGRAM:

```
#include <stdio.h>
#include <limits.h>
int matrixChainOrder(int p[], int n) {
    int m[n][n];
```

```
    for (int i = 1; i < n; i++) {
        m[i][i] = 0;
    }
    for (int L = 2; L < n; L++) {
        for (int i = 1; i < n - L + 1; i++) {
            int j = i + L - 1;
            m[i][j] = INT_MAX;
            for (int k = i; k < j; k++) {
                int cost = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                if (cost < m[i][j]) {
                    m[i][j] = cost;
                }
            }
        }
    }
    return m[1][n - 1];
}

int main() {
    int n;
    printf("Enter the number of matrices: ");
    scanf("%d", &n);
    int p[n + 1];
    printf("Enter the dimensions of the matrices (array of size %d):\n", n + 1);
    for (int i = 0; i <= n; i++) {
        scanf("%d", &p[i]);
    }
    int cost = matrixChainOrder(p, n + 1);
    printf("Minimum number of scalar multiplications is: %d\n", cost);

    return 0;
}
```

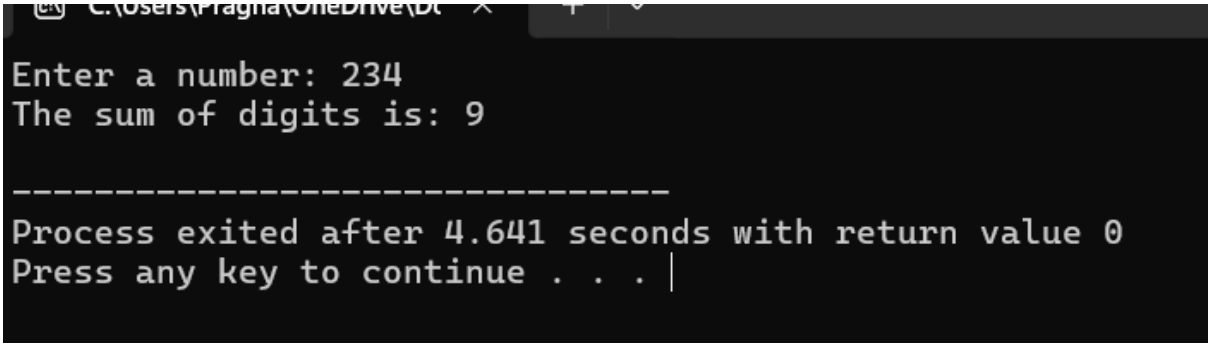
OUTPUT:



30.WRITE A PROGRAM FOR SUM OF DIGITS
A
PROGRAM:

```
#include <stdio.h>
int sumOfDigits(int num) {
    int sum = 0;
    while (num != 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num < 0) {
        num = -num;
    }
    int result = sumOfDigits(num);
    printf("The sum of digits is: %d\n", result);

    return 0;
}
```

OUTPUT:

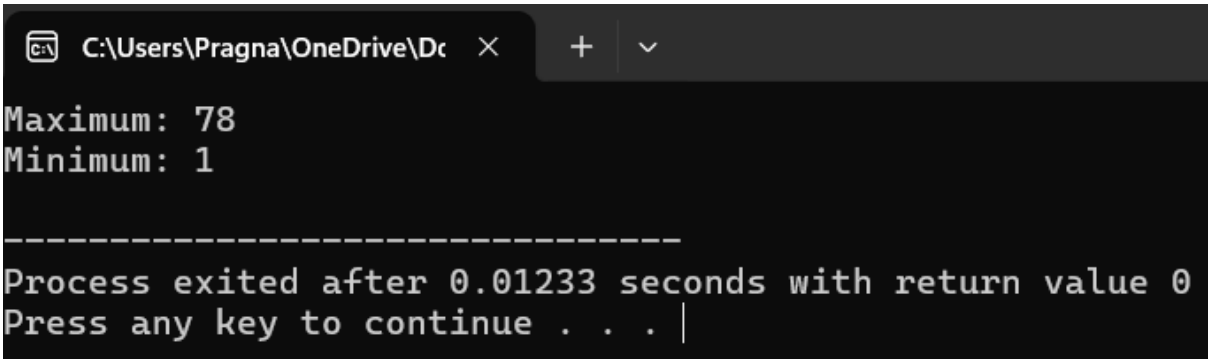


31.WRITE A PROGRAM TO PRINT MIN AND MAX VALUES SEQUENCY FOR ALL THE NUMBERS IN THE LIST
A
PROGRAM:

```
#include <stdio.h>
void findMinMax(int arr[], int n) {
    int min = arr[0], max = arr[0];
    int minPos = 0, maxPos = 0;
    for (int i = 1; i < n; i++) {
        if (arr[i] < min) {
            min = arr[i];
            minPos = i;
        }
        if (arr[i] > max) {
```

```
            max = arr[i];
            maxPos = i;
        }
    }
    printf("Minimum Value: %d at position %d\n", min, minPos + 1);
    printf("Maximum Value: %d at position %d\n", max, maxPos + 1);
}
int main() {
    int n;
    printf("Enter the number of elements in the list: ");
    scanf("%d", &n);
          int arr[n];
    printf("Enter the elements of the list:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    findMinMax(arr, n);
          return 0;
}
```

OUTPUT:



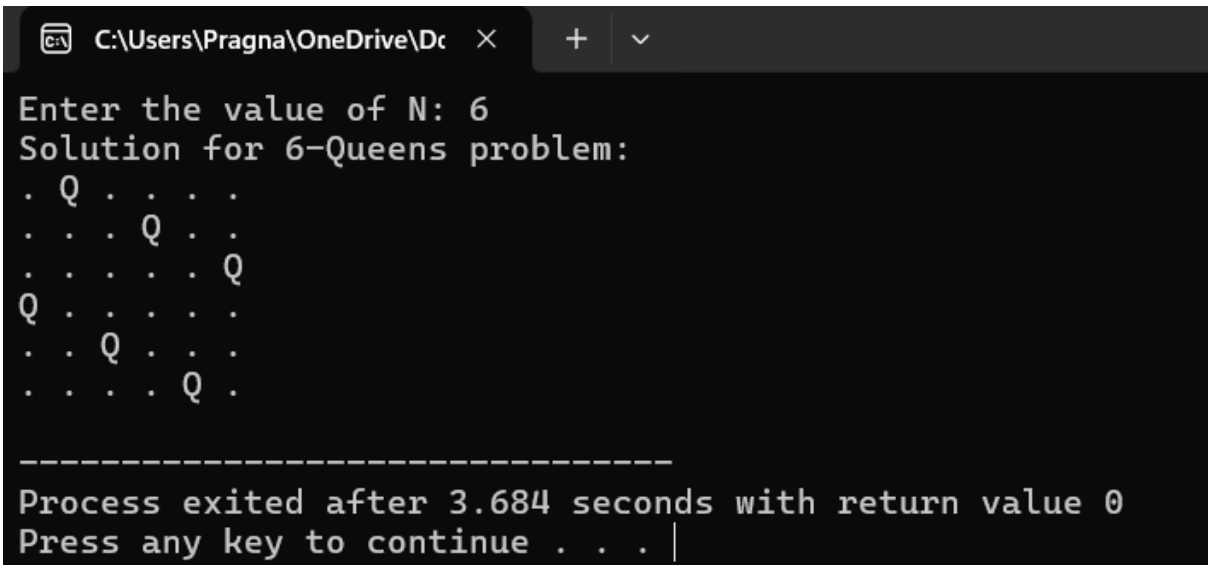
32.WRITE A PROGRAM FOR N QUEEN PROBLEM USING BACKTRACKING
A
PROGRAM:

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 10
int board[MAX][MAX];
bool isSafe(int n, int row, int col) {
    int i, j;
    for (i = 0; i < row; i++) {
        if (board[i][col]) {
            return false;
        }
    }
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j]) {
            return false;
        }
```

```
    }
    for (i = row, j = col; i >= 0 && j < n; i--, j++) {
        if (board[i][j]) {
            return false;
        }
    }

    return true;
}
bool solveNQueens(int n, int row) {
    if (row == n) {
        return true;
    }
    for (int col = 0; col < n; col++) {
        if (isSafe(n, row, col)) {
            board[row][col] = 1;
            if (solveNQueens(n, row + 1)) {
                return true;
            }
                            board[row][col] = 0;
        }
    }
          return false;
}
void printSolution(int n) {
    printf("Solution for %d-Queens problem:\n", n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%c ", board[i][j] ? 'Q' : '.');
        }
        printf("\n");
    }
}

int main() {
    int n;
    printf("Enter the value of N: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            board[i][j] = 0;
        }
    }
    if (solveNQueens(n, 0)) {
        printSolution(n);
    } else {
        printf("No solution exists for %d-Queens problem.\n", n);
    }

    return 0;
```

}

OUTPUT:

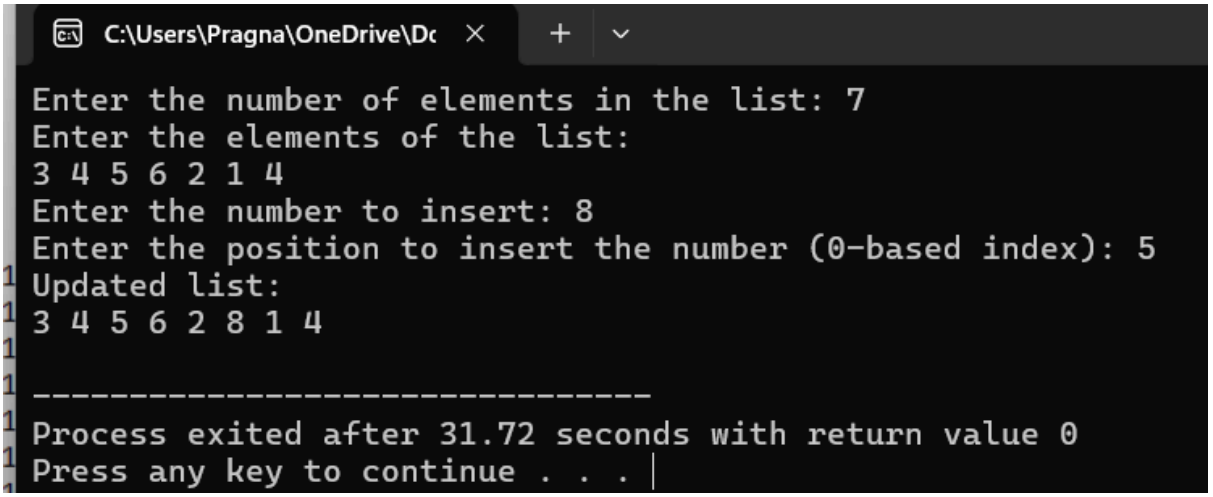


33.WRITE A PROGRAM TO INSERT A NUMBER IN A LIST
A
PROGRAM:

```
#include <stdio.h>
void insertNumber(int arr[], int *n, int num, int pos) {
    for (int i = *n; i > pos; i--) {
        arr[i] = arr[i - 1];
    }
    arr[pos] = num;
    (*n)++;
}
int main() {
    int arr[100], n, num, pos;
    printf("Enter the number of elements in the list: ");
    scanf("%d", &n);
    printf("Enter the elements of the list:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the number to insert: ");
    scanf("%d", &num);
    printf("Enter the position to insert the number (0-based index): ");
    scanf("%d", &pos);
    if (pos < 0 || pos > n) {
        printf("Invalid position! Please enter a position between 0 and %d.\n", n);
        return 1;
    }
    insertNumber(arr, &n, num, pos);
    printf("Updated list:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
```

```
    }
    printf("\n");

    return 0;
}
```

OUTPUT:



34.WRITE A PROGRAM TO PERFORM SUM OF SUBSETS USING BACKTRACKING

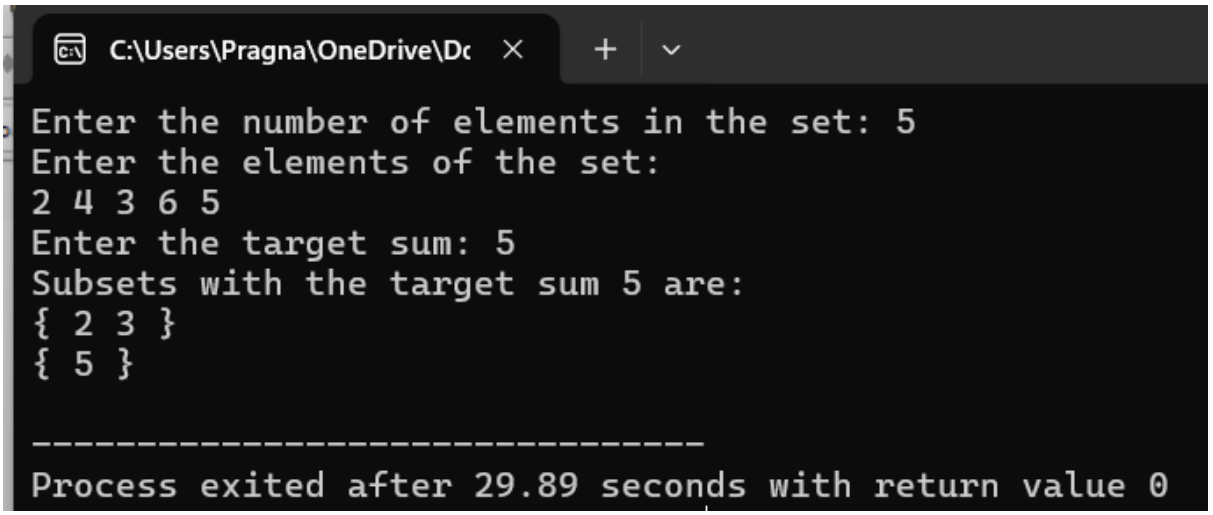
A

PROGRAM:

```
#include <stdio.h>
#define MAX 100
int subset[MAX];
void printSubset(int subset[], int size) {
    printf("{ ");
    for (int i = 0; i < size; i++) {
        printf("%d ", subset[i]);
    }
    printf("}\n");
}
void sumOfSubsets(int arr[], int n, int targetSum, int currentSum, int index, int subsetSize) {
    if (currentSum == targetSum) {
        printSubset(subset, subsetSize);
        return;
    }
    if (currentSum > targetSum || index == n) {
        return;
    }
    subset[subsetSize] = arr[index];
        sumOfSubsets(arr, n, targetSum, currentSum + arr[index], index + 1, subsetSize + 1);
    sumOfSubsets(arr, n, targetSum, currentSum, index + 1, subsetSize);
}
int main() {
```

```
    int n, targetSum;
    printf("Enter the number of elements in the set: ");
    scanf("%d", &n);
        int arr[n];
    printf("Enter the elements of the set:\n");
    for (int i = 0; i < n; i++) {
       scanf("%d", &arr[i]);
    }
    printf("Enter the target sum: ");
    scanf("%d", &targetSum);
    printf("Subsets with the target sum %d are:\n", targetSum);
        sumOfSubsets(arr, n, targetSum, 0, 0, 0);
        return 0;
}
```

OUTPUT:



35.WRITE A PROGRAM TO PERFORM GRAPH COLORING USIING BACKTRACKING METHOD

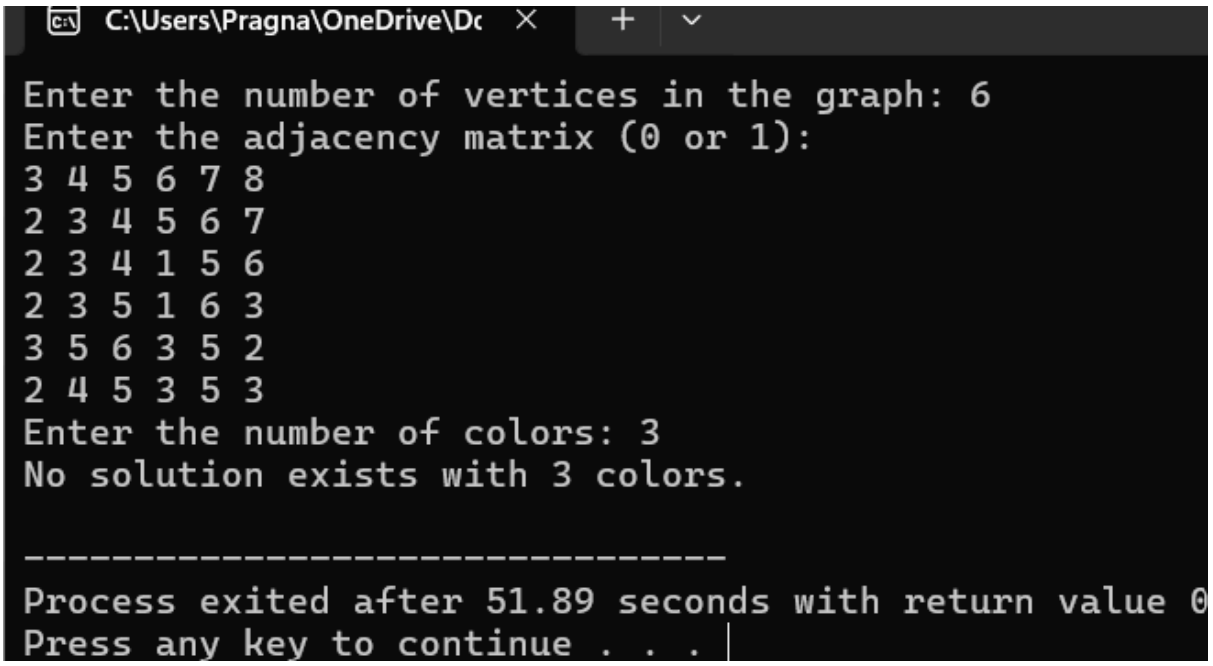
A

PROGRAM:

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 100
int graph[MAX][MAX];
int colors[MAX];
bool isSafe(int v, int c, int V) {
    for (int i = 0; i < V; i++) {
        if (graph[v][i] && colors[i] == c) {
            return false;
        }
    }
    return true;
}
bool graphColoringUtil(int V, int m, int v) {
    if (v == V) {
```

```
        return true;
    }
    for (int c = 1; c <= m; c++) {
        if (isSafe(v, c, V)) {
            colors[v] = c;
            if (graphColoringUtil(V, m, v + 1)) {
                return true;
            }

            colors[v] = 0;
        }
    }
    return false;
}
bool graphColoring(int V, int m) {
    for (int i = 0; i < V; i++) {
        colors[i] = 0;
    }
    if (graphColoringUtil(V, m, 0)) {
        printf("Solution exists with %d colors:\n", m);
        for (int i = 0; i < V; i++) {
            printf("Vertex %d -> Color %d\n", i, colors[i]);
        }
        return true;
    }

    printf("No solution exists with %d colors.\n", m);
    return false;
}
int main() {
    int V, m;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &V);
    printf("Enter the adjacency matrix (0 or 1):\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            scanf("%d", &graph[i][j]);
            if (i != j && graph[i][j] != graph[j][i]) {
                graph[j][i] = graph[i][j];
            }
        }
    }
    printf("Enter the number of colors: ");
    scanf("%d", &m);
    graphColoring(V, m);

    return 0;
}
```

OUTPUT:

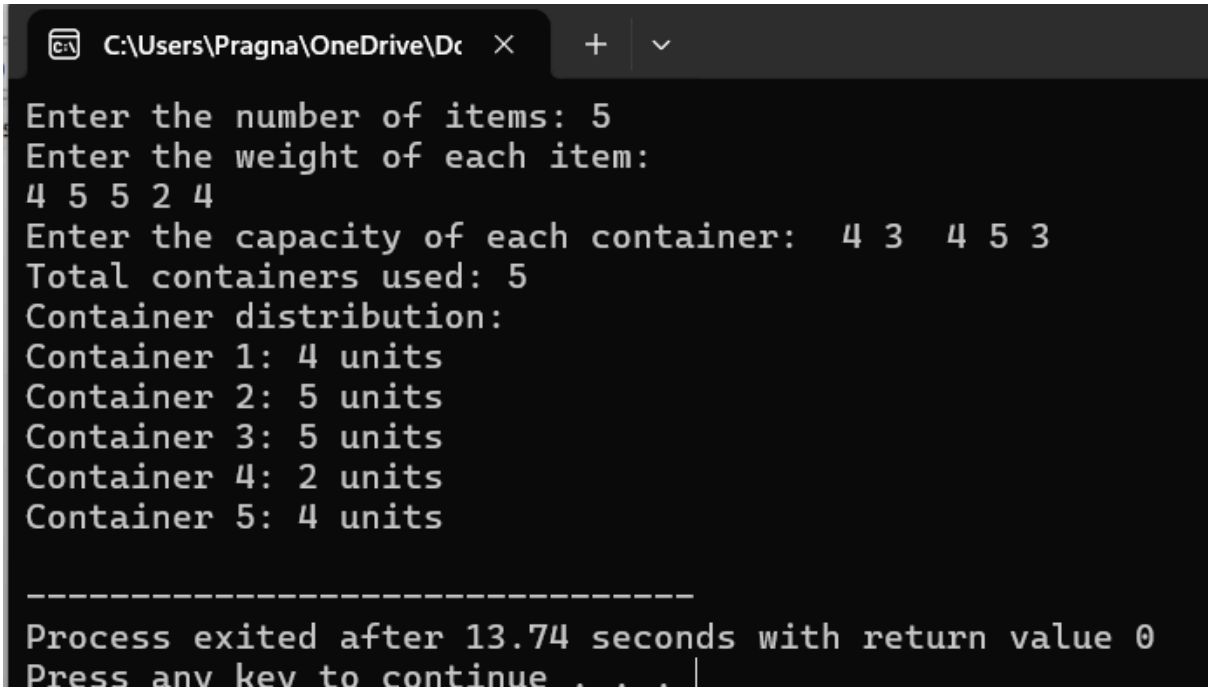


36.WRITE A PROGRAM TO COMPUTE CONTAINER LOADER PROBLEM
A
PROGRAM:

```
#include <stdio.h>
#define MAX 100
void firstFit(int items[], int n, int capacity) {
    int containers[MAX];
    int containerCount = 0;
    for (int i = 0; i < MAX; i++) {
        containers[i] = 0;
    }
    for (int i = 0; i < n; i++) {
        int placed = 0;
        for (int j = 0; j < containerCount; j++) {
            if (containers[j] + items[i] <= capacity) {
                containers[j] += items[i];
                placed = 1;
                break;
            }
        }
        if (!placed) {
            containers[containerCount] = items[i];
            containerCount++;
        }
    }
    printf("Total containers used: %d\n", containerCount);
    printf("Container distribution:\n");
    for (int i = 0; i < containerCount; i++) {
        printf("Container %d: %d units\n", i + 1, containers[i]);
    }
}
```

```
int main() {
    int n, capacity;
    printf("Enter the number of items: ");
    scanf("%d", &n);
          int items[n];
    printf("Enter the weight of each item:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &items[i]);
    }
    printf("Enter the capacity of each container: ");
    scanf("%d", &capacity);
    firstFit(items, n, capacity);

    return 0;
}
```

OUTPUT:



37.WRITE A PROGRAM TO GENERATE A LIST OF FACTORS FOR A NUMBER N VALUE

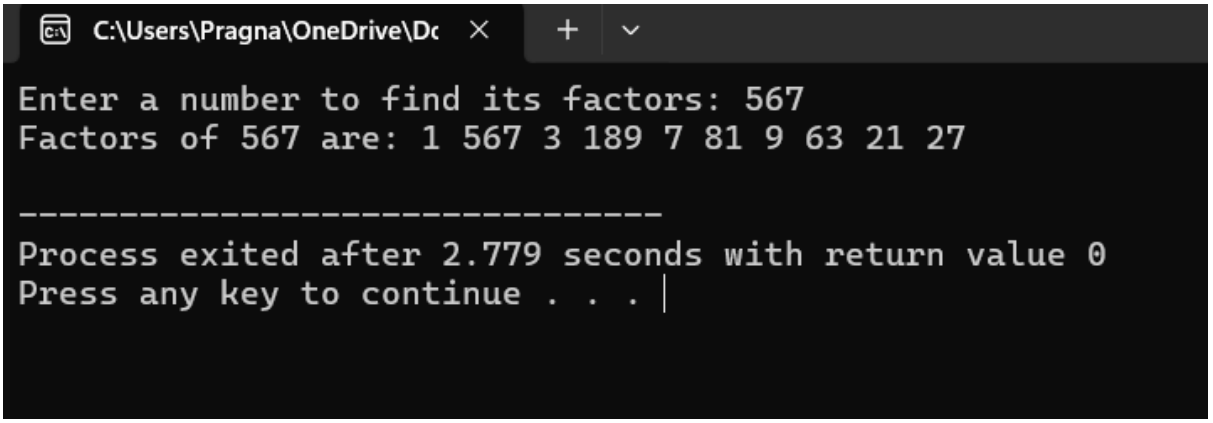
A

PROGRAM:

```
#include <stdio.h>
void findFactors(int n) {
    if (n <= 0) {
        printf("Number must be greater than 0.\n");
        return;
    }
          printf("Factors of %d are: ", n);
    for (int i = 1; i * i <= n; i++) {
        if (n % i == 0) {
```

```
            printf("%d ", i);
            if (i != n / i) {
                printf("%d ", n / i);
            }
        }
    }
    printf("\n");
}
int main() {
    int number;
          printf("Enter a number to find its factors: ");
    if (scanf("%d", &number) != 1) {
        printf("Invalid input. Please enter an integer.\n");
        return 1;
    }

    findFactors(number);

    return 0;
}
```

OUTPUT:



38.WRITE A PROGRAM USING ASSIGNMENT PROBLEM USING BRANCH AND BOUND

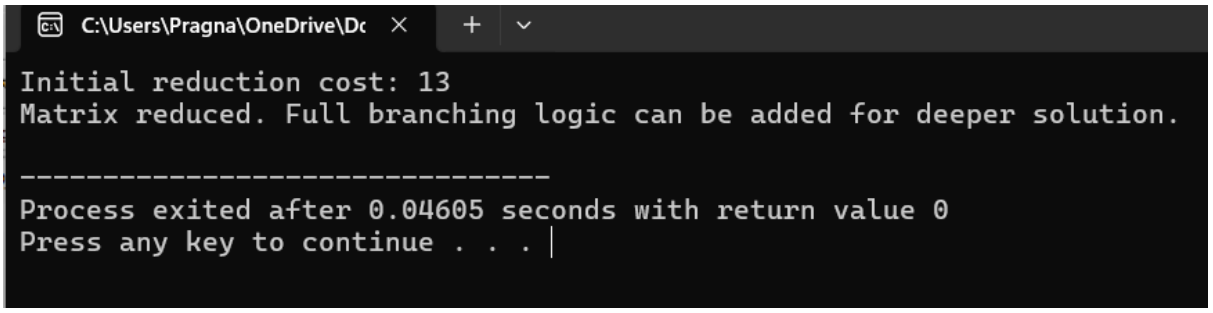
A

PROGRAM:

```
#include <stdio.h>
#include <limits.h>
#define N 4
int reduceMatrix(int cost[N][N], int n) {
    int reduction = 0;
    for (int i = 0; i < n; i++) {
        int rowMin = INT_MAX;
        for (int j = 0; j < n; j++)
            if (cost[i][j] < rowMin)
                rowMin = cost[i][j];
        reduction += rowMin;
        for (int j = 0; j < n; j++)
```

```
            cost[i][j] -= rowMin;
    }
    for (int j = 0; j < n; j++) {
        int colMin = INT_MAX;
        for (int i = 0; i < n; i++)
            if (cost[i][j] < colMin)
                colMin = cost[i][j];
        reduction += colMin;
        for (int i = 0; i < n; i++)
            cost[i][j] -= colMin;
    }
          return reduction;
}
int main() {
    int cost[N][N] = {
        {9, 2, 7, 8},
        {6, 4, 3, 7},
        {5, 8, 1, 8},
        {7, 6, 9, 4}
    };
          int totalCost = reduceMatrix(cost, N);
    printf("Initial reduction cost: %d\n", totalCost);
    printf("Matrix reduced. Full branching logic can be added for deeper solution.\n");

    return 0;
}
```

OUTPUT:

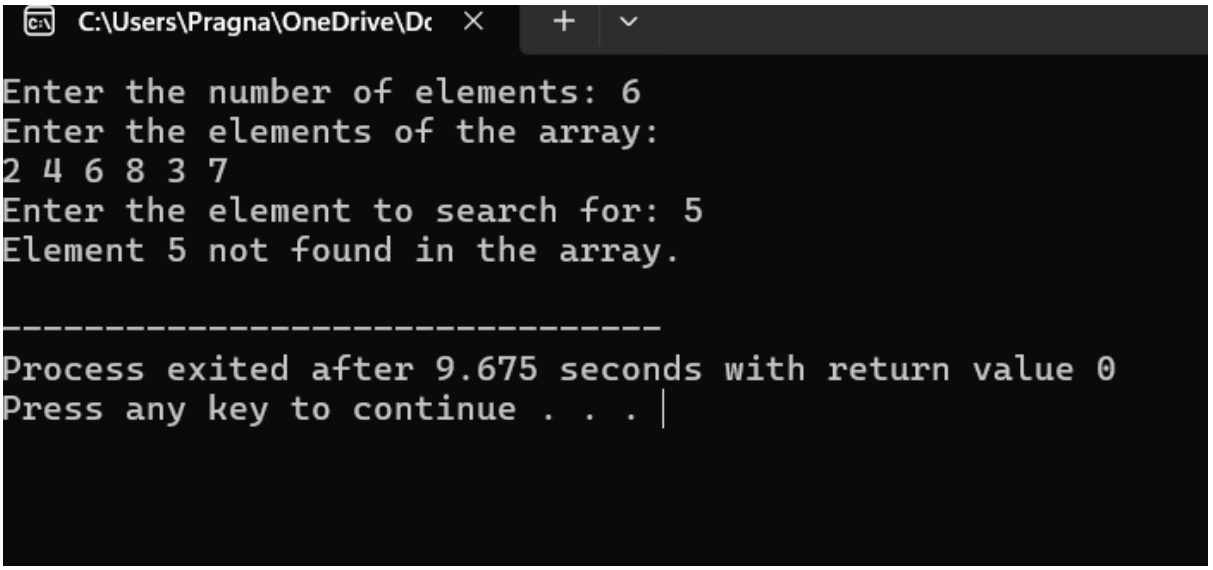


39.WRITE A PROGRAM TO PERFORM LINEAR SEARCH
A
PROGRAM:

```
#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 n, target;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
          int arr[n];
    printf("Enter the elements of the array: \n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search for: ");
    scanf("%d", &target);
    int result = linearSearch(arr, n, target);
    if (result != -1) {
        printf("Element %d found at index %d.\n", target, result);
    } else {
        printf("Element %d not found in the array.\n", target);
    }
          return 0;
}
```

RESULT:



40.WRITE A PROGRAM TO FIND HAMILTON PROBLEM USING BACKTRACKING METHOD

A

PROGRAM:

```
#include <stdio.h>
#include <stdbool.h>
#define V 5
bool isSafe(int v, int graph[V][V], int path[], int pos) {
    if (graph[path[pos - 1]][v] == 0) {
        return false;
}
    for (int i = 0; i < pos; i++) {
        if (path[i] == v) {
            return false;
```

```
        }
    }
        return true;
}
bool hamiltonianCycleUtil(int graph[V][V], int path[], int pos) {
    if (pos == V) {
        if (graph[path[pos - 1]][path[0]] == 1) {
            return true;
        }
        return false;
}
    for (int v = 1; v < V; v++) {
        if (isSafe(v, graph, path, pos)) {
            path[pos] = v;

            if (hamiltonianCycleUtil(graph, path, pos + 1)) {
                return true;
            }
            path[pos] = -1;
        }
    }
        return false;
}
bool hamiltonianCycle(int graph[V][V]) {
    int path[V];
    for (int i = 0; i < V; i++) {
        path[i] = -1;
    }
    path[0] = 0;
        if (hamiltonianCycleUtil(graph, path, 1)) {
        printf("Hamiltonian Cycle: ");
        for (int i = 0; i < V; i++) {
            printf("%d ", path[i]);
        }
        printf("\n");
        return true;
    } else {
        printf("No solution exists\n");
        return false;
    }
}
int main() {
int graph[V][V] = {
        {0, 1, 0, 1, 0},
        {1, 0, 1, 1, 0},
        {0, 1, 0, 1, 1},
        {1, 1, 1, 0, 1},
        {0, 0, 1, 1, 0}
    };
    hamiltonianCycle(graph);
```

```
    return 0;
}
```

OUTPUT