CSA0699 – DESIGN AND ANALYSIS OF ALGORITHMS FOR OPTIMAL METHODS

LAB PROGRAMS (40)

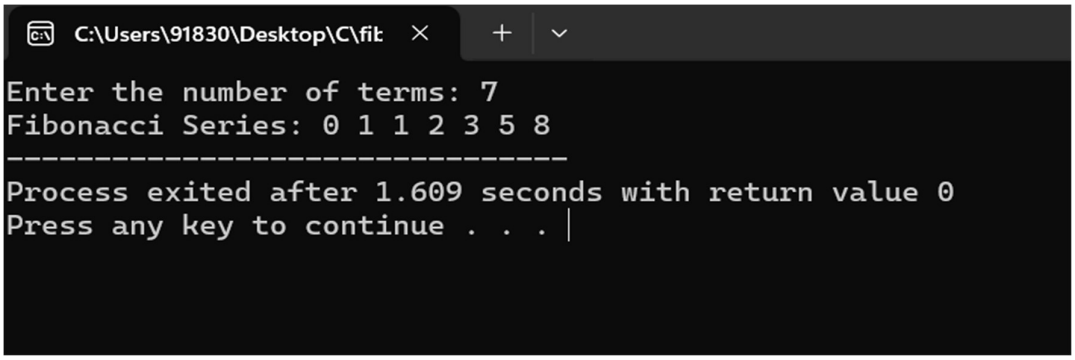
M. RAVINDRANATH REDDY

192211942

1)Write a program to Print Fibonacci Series using recursion.

```
#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: ");
scanf("%d", &n);
printf("Fibonacci Series: ");
for (i = 0; i < n; i++) {
printf("%d ", fibonacci(i));
}
return 0;
}
```

Output

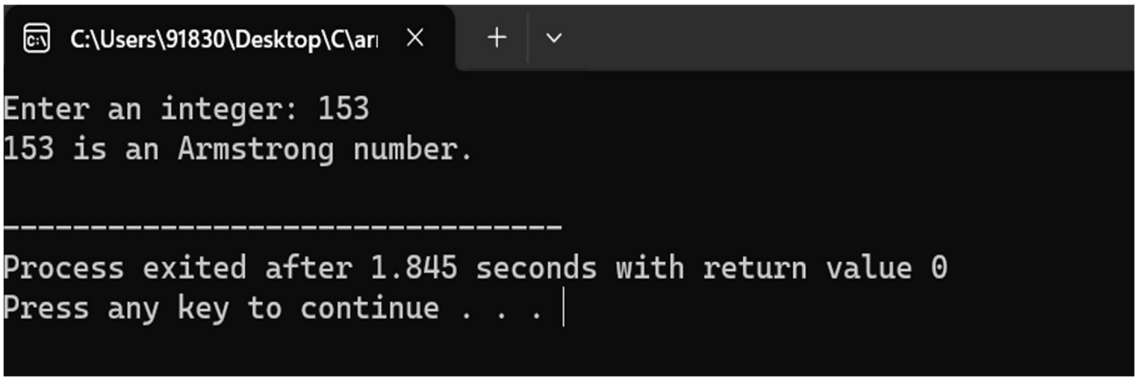


2) Write a program to check the given no is Armstrong or not.

```
#include <stdio.h>
#include <math.h>
int main() {
    int num, originalNum, remainder, result = 0, n = 0;
```

```
    printf("Enter an integer: ");
    scanf("%d", &num);

    originalNum = num;

    while (originalNum != 0) {
        originalNum /= 10;
        ++n;
    }

    originalNum = num;
    while (originalNum != 0) {
        remainder = originalNum % 10;
        result += pow(remainder, n);
        originalNum /= 10;
    }
    if (result == num) {
        printf("%d is an Armstrong number.\n", num);
    } else {
        printf("%d is not an Armstrong number.\n", num);
    }
    return 0;
}
```

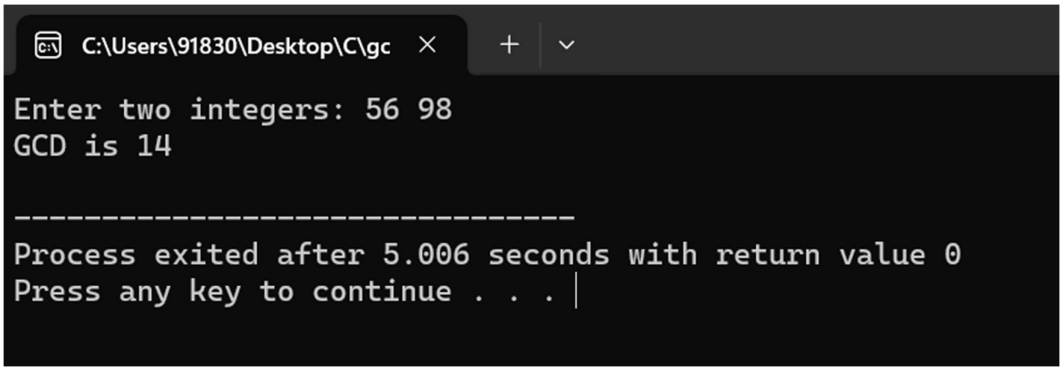
Output



3) Program to find the GCD of two numbers .

```
#include <stdio.h>

int main() {
    int a, b, gcd;

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

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

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

Output

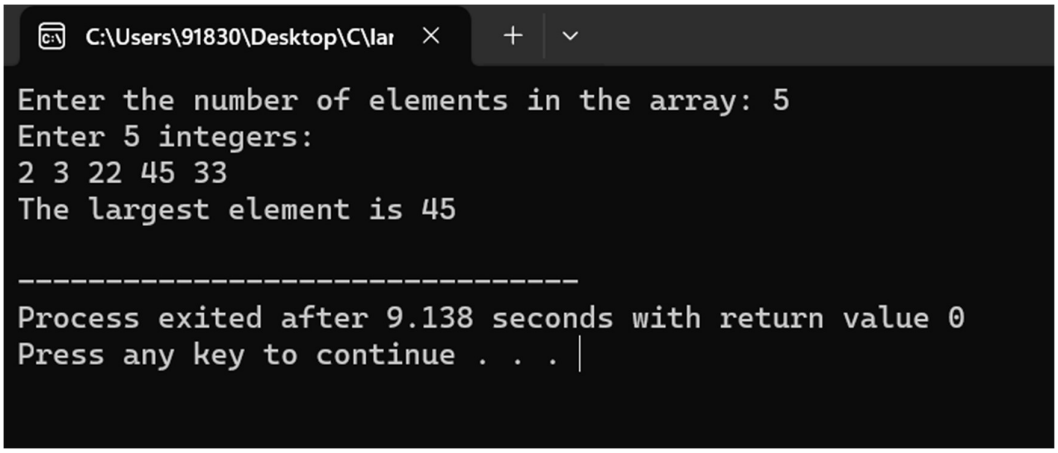


4) Write a program to get the largest element of an array.

```
#include <stdio.h>

int main() {
    int n, i, largest;
```

```
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int arr[n];

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

    largest = arr[0];

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

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

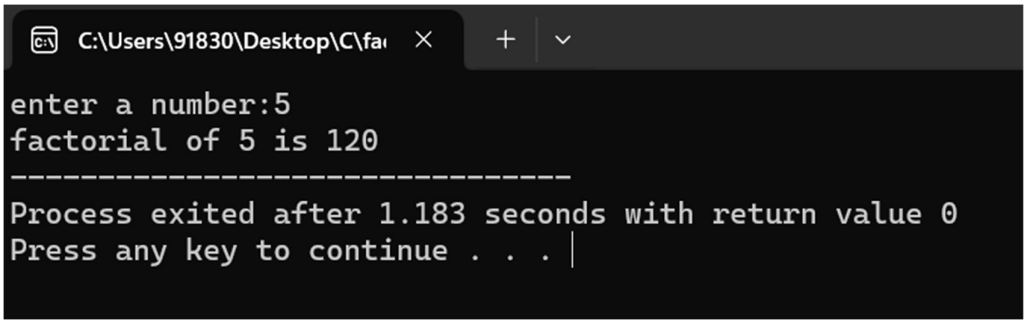
Output



5) Write a program to find the Factorial of a number

```
#include<stdio.h>
int main(){
        int n;
        int fact=1;
        printf("enter a number:");
        scanf("%d",&n);
        for(int i=1;i<=n;i++){
                fact=fact*i;
        }
        printf("factorial of %d is %d",n,fact);
        return 0;
}
```

Output

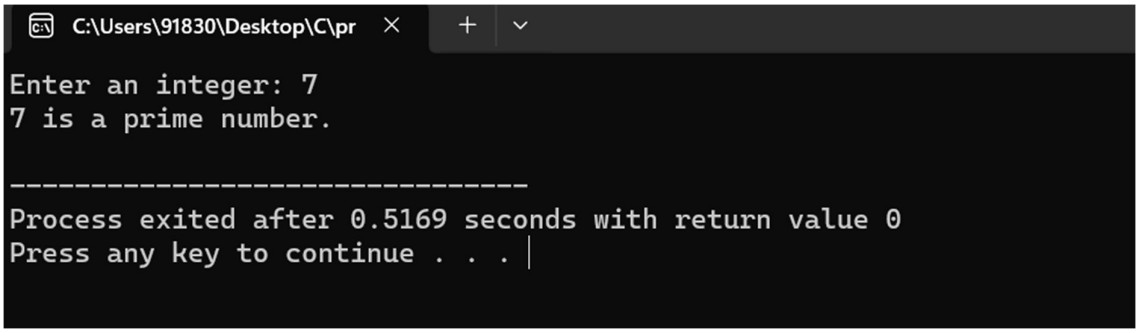


6) Write a program to check a number is a prime number or not .

```
#include <stdio.h>

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

    printf("Enter an integer: ");
    scanf("%d", &num);

    if (num <= 1) {
        isPrime = 0;
```

```
    } else {
        for (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



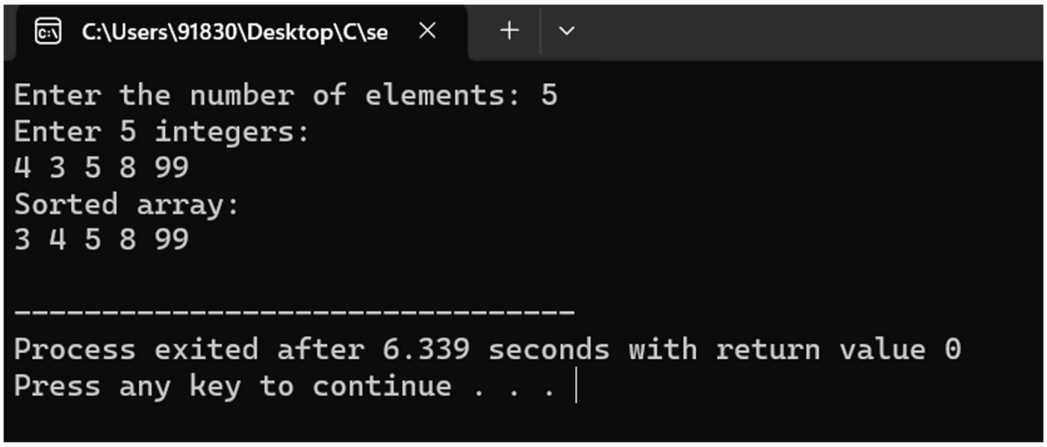
7) Write a program to perform Selection sort.

```
#include <stdio.h>

void selectionSort(int arr[], int n) {
    int i, j, minIndex, temp;

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

```
        minIndex = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}

int main() {
    int n, i;

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

    int arr[n];

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

    selectionSort(arr, n);

    printf("Sorted array:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
```

```
    return 0;
}
```

Ouptut

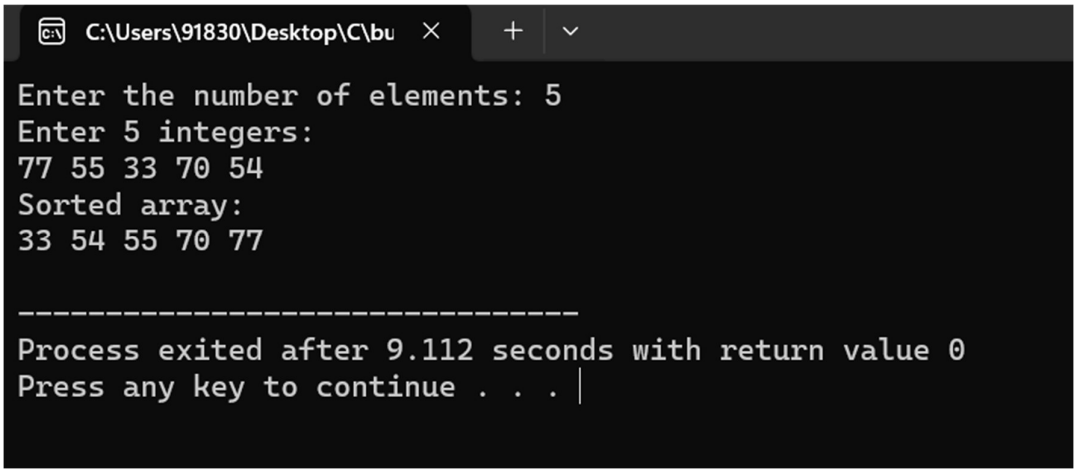


8) Write a program to perform Bubble sort

```
#include <stdio.h>

void bubbleSort(int arr[], int n) {
    int i, j, temp;
    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;
            }
        }
    }
}

int main() {
    int n, i;

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

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

    int arr[n];

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

    bubbleSort(arr, n);

    printf("Sorted array:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Output



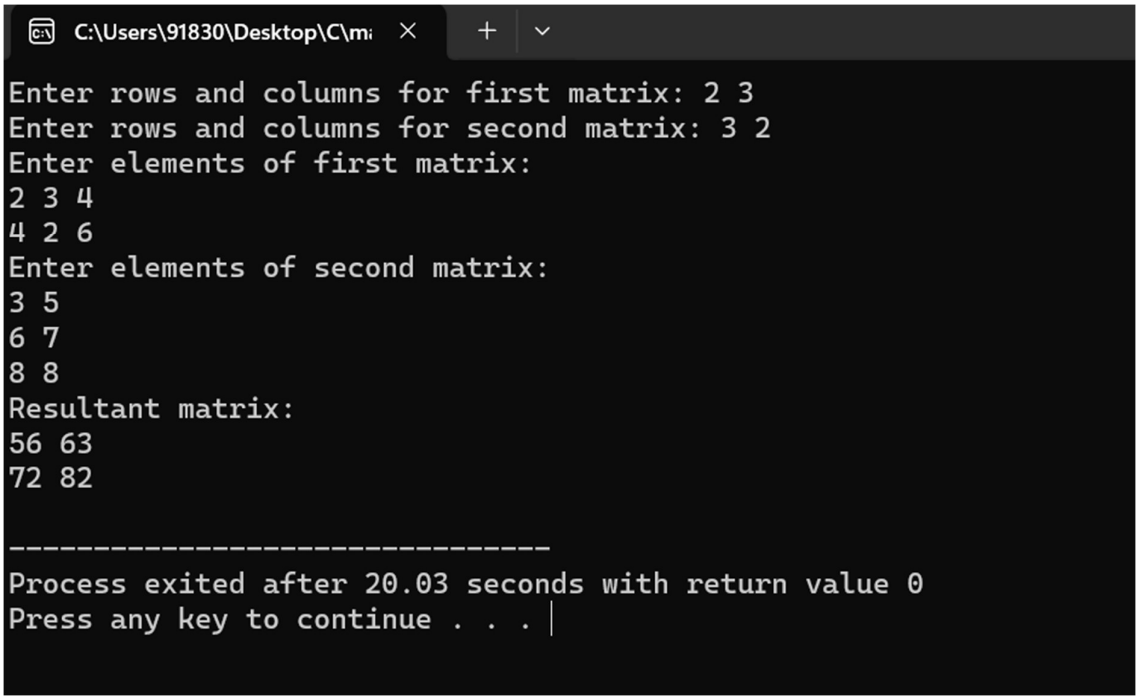
9) Write a program for to multiply two Matrix

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

```
#define MAX 10

int main() {
    int a[MAX][MAX], b[MAX][MAX], result[MAX][MAX];
    int r1, c1, r2, c2, i, j, k;

    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 is not possible.\n");
        return 1;
    }

    printf("Enter elements of first matrix:\n");
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c1; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    printf("Enter elements of second matrix:\n");
    for (i = 0; i < r2; i++) {
        for (j = 0; j < c2; j++) {
            scanf("%d", &b[i][j]);
        }
    }
    for (i = 0; i < r1; i++) {
```

```
    for (j = 0; j < c2; j++) {
            result[i][j] = 0;
            for (k = 0; k < c1; k++) {
                result[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    printf("Resultant matrix:\n");
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c2; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

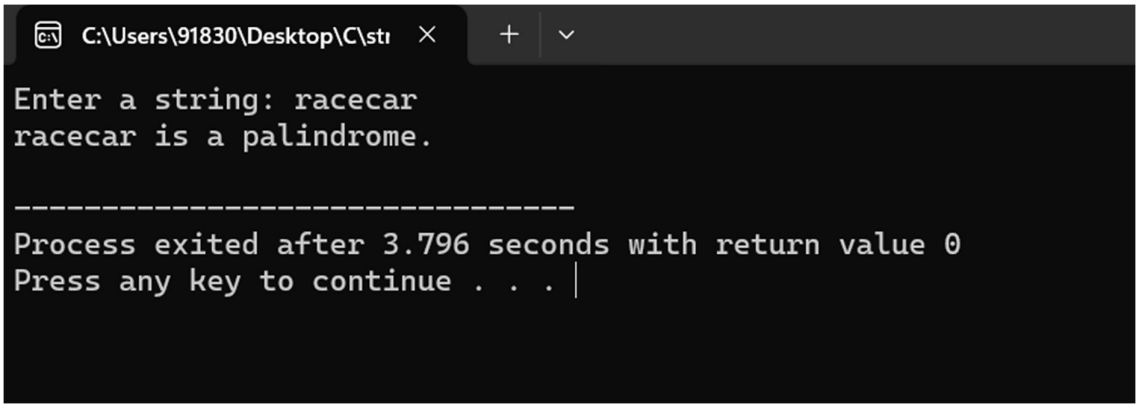
Output



10) Write a program for to check whether a given String is Palindrome or not

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[100], reversed[100];
    int length, i, j;
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = 0;
    length = strlen(str);
    j = 0;
    for (i = length - 1; i >= 0; i--) {
        reversed[j++] = str[i];
    }
    reversed[j] = '\0';
    if (strcmp(str, reversed) == 0) {
        printf("%s is a palindrome.\n", str);
    } else {
        printf("%s is not a palindrome.\n", str);
    }
    return 0;
}
```

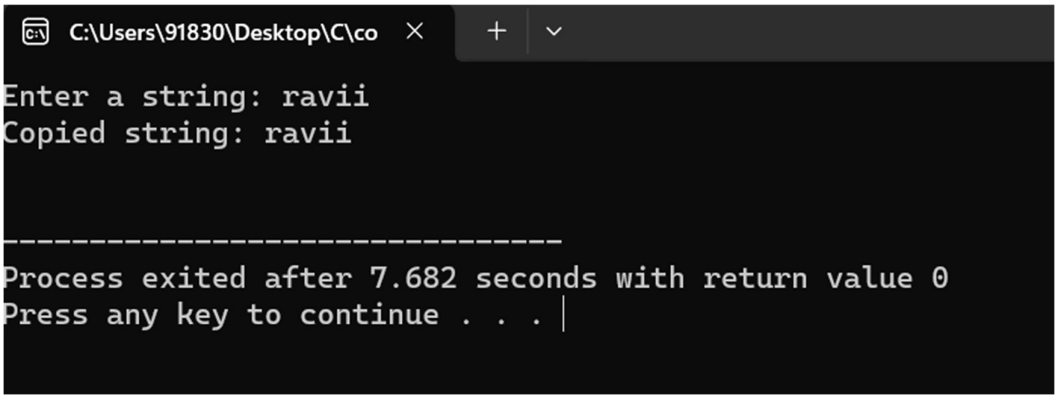
Output



11) Write a program for to copy one string to another

```
#include <stdio.h>
int main() {
    char source[100], destination[100];
    int i;
    printf("Enter a string: ");
    fgets(source, sizeof(source), stdin);
    for (i = 0; source[i] != '\0'; i++) {
        destination[i] = source[i];
    }
    destination[i] = '\0';
    printf("Copied string: %s\n", destination);
    return 0;
}
```

Output

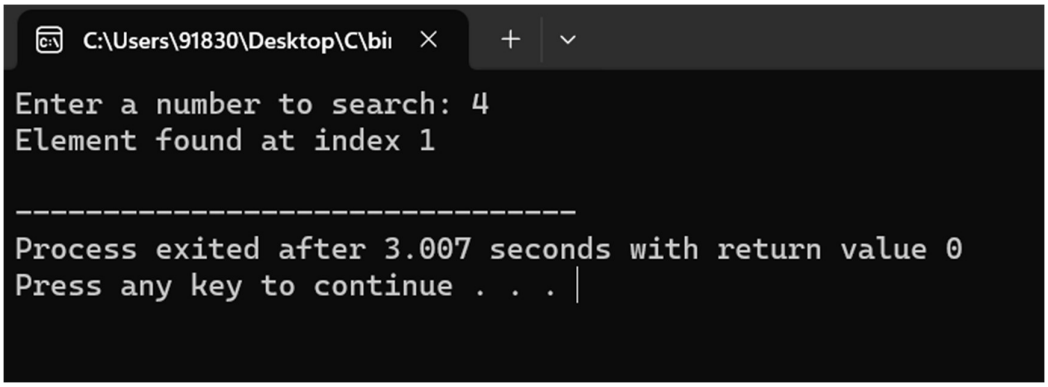


12) Write a Program to perform binary search.

```
#include <stdio.h>
int binarySearch(int arr[], int size, int target) {
    int left = 0, right = size - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return mid;
        }
```

```
  if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1;
}
int main() {
    int arr[] = {2, 4, 6, 8, 10, 12, 14, 16, 18};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target, result;
    printf("Enter a number to search: ");
    scanf("%d", &target);
    result = binarySearch(arr, size, target);
    if (result != -1) {
        printf("Element found at index %d\n", result);
    } else {
        printf("Element not found\n");
    }
    return 0;
}
```

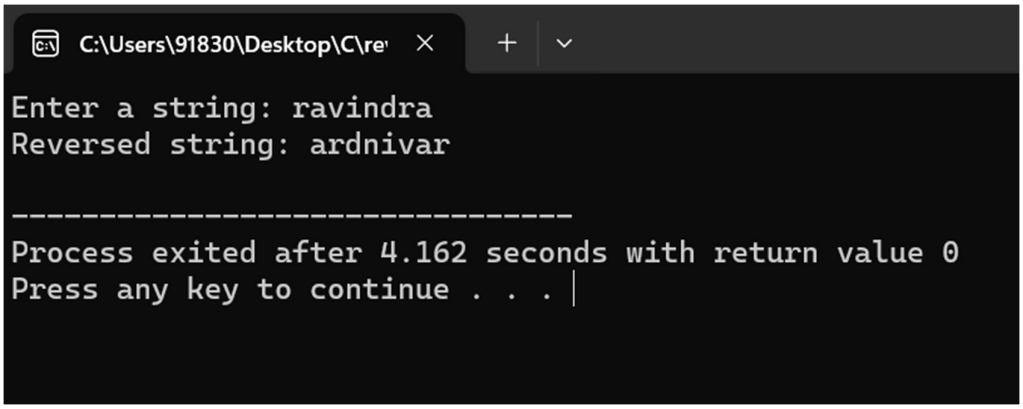
Output



13) Write a program to print the reverse of a string

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[100];
    int length, i;
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = 0;
    length = strlen(str);
    printf("Reversed string: ");
    for (i = length - 1; i >= 0; i--) {
        putchar(str[i]);
    }
    printf("\n");
    return 0;
}
```

Output

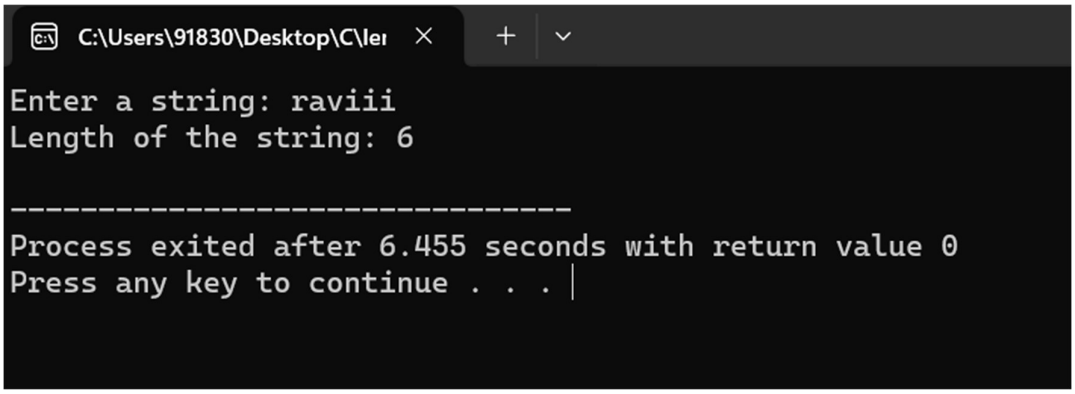


14) Write a program to find the length of a string.

```
#include <stdio.h>
int main() {
    char str[100];
    int length = 0;
```

```
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = 0;
    while (str[length] != '\0') {
        length++;
    }
    printf("Length of the string: %d\n", length);
    return 0;
}
```

Output



15) Write a program to perform Strassen's Matrix Multiplication.

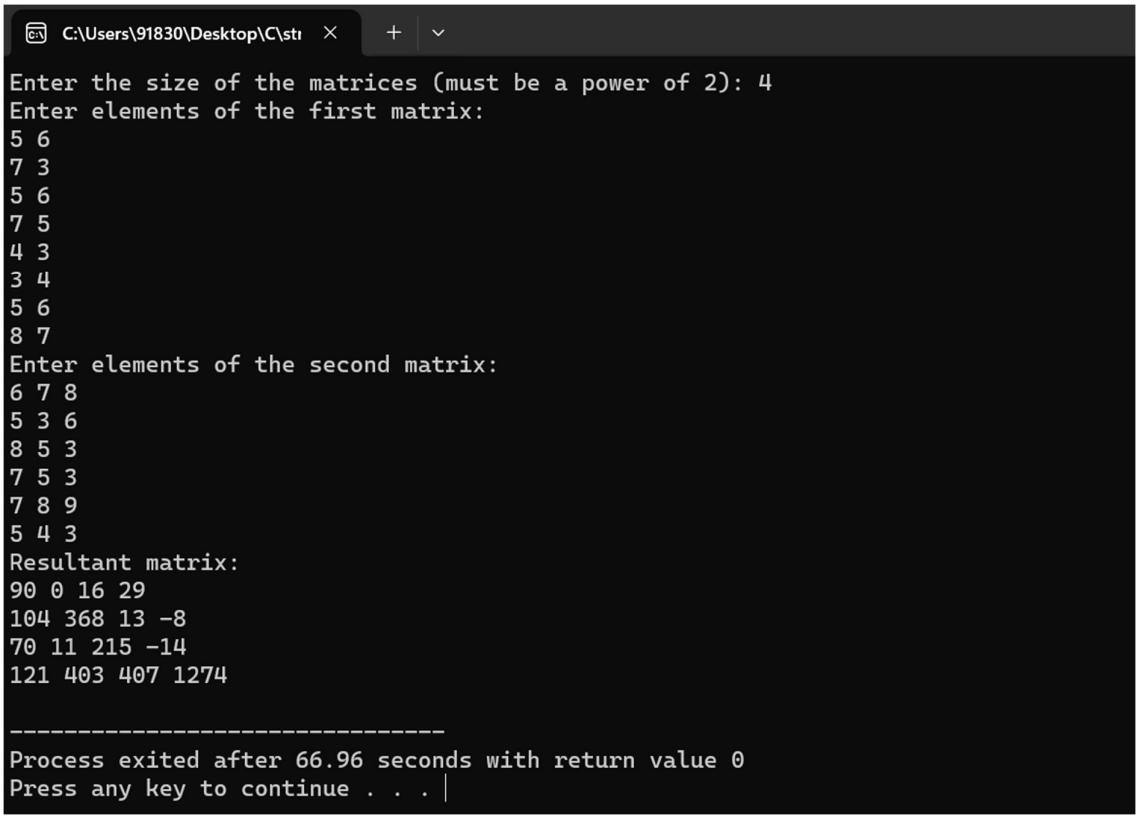
```
#include <stdio.h>
#include <stdlib.h>
void addMatrix(int A[10][10], int B[10][10], int C[10][10], int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] + B[i][j];
}
void subtractMatrix(int A[10][10], int B[10][10], int C[10][10], int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] - B[i][j];
}
```

```
void strassen(int A[10][10], int B[10][10], int C[10][10], int n) {
    if (n == 1) {
        C[0][0] = A[0][0] * B[0][0];
        return;
    }
    int k = n / 2;
    int A11[10][10], A12[10][10], A21[10][10], A22[10][10];
    int B11[10][10], B12[10][10], B21[10][10], B22[10][10];
    int M1[10][10], M2[10][10], M3[10][10], M4[10][10], M5[10][10], M6[10][10], M7[10][10];
    int C11[10][10], C12[10][10], C21[10][10], C22[10][10];
    int temp1[10][10], temp2[10][10];
    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + k];
            A21[i][j] = A[i + k][j];
            A22[i][j] = A[i + k][j + k];
            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + k];
            B21[i][j] = B[i + k][j];
            B22[i][j] = B[i + k][j + k];
        }
    addMatrix(A11, A22, temp1, k);
    addMatrix(B11, B22, temp2, k);
    strassen(temp1, temp2, M1, k);
    addMatrix(A21, A22, temp1, k);
    strassen(temp1, B11, M2, k);
    subtractMatrix(B12, B22, temp2, k);
    strassen(A11, temp2, M3, k);
    subtractMatrix(B21, B11, temp2, k);
    strassen(A22, temp2, M4, k);
    addMatrix(A11, A12, temp1, k);
```

```
    strassen(temp1, B22, M5, k);
    subtractMatrix(A21, A11, temp1, k);
    addMatrix(B11, B12, temp2, k);
    strassen(temp1, temp2, M6, k);
    subtractMatrix(A12, A22, temp1, k);
    addMatrix(B21, B22, temp2, k);
    strassen(temp1, temp2, M7, k);
    addMatrix(M1, M4, C11, k);
    subtractMatrix(C11, M5, C11, k);
    addMatrix(C11, M7, C11, k);
    addMatrix(M3, M4, C12, k);
    addMatrix(M2, M4, C21, k);
    addMatrix(M1, M2, C22, k);
    subtractMatrix(C22, M3, C22, k);
    addMatrix(C22, M6, C22, k);
    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
          C[i][j] = C11[i][j];
           C[i][j + k] = C12[i][j];
           C[i + k][j] = C21[i][j];
           C[i + k][j + k] = C22[i][j];
        }
}
int main() {
    int n;
    printf("Enter the size of the matrices (must be a power of 2): ");
    scanf("%d", &n);
    int A[10][10], B[10][10], C[10][10];
    printf("Enter elements of the first matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &A[i][j]);
```

```
    printf("Enter elements of the second matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &B[i][j]);
    strassen(A, B, C, n);
    printf("Resultant matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            printf("%d ", C[i][j]);
        printf("\n");
    }
    return 0;
}
```

Output



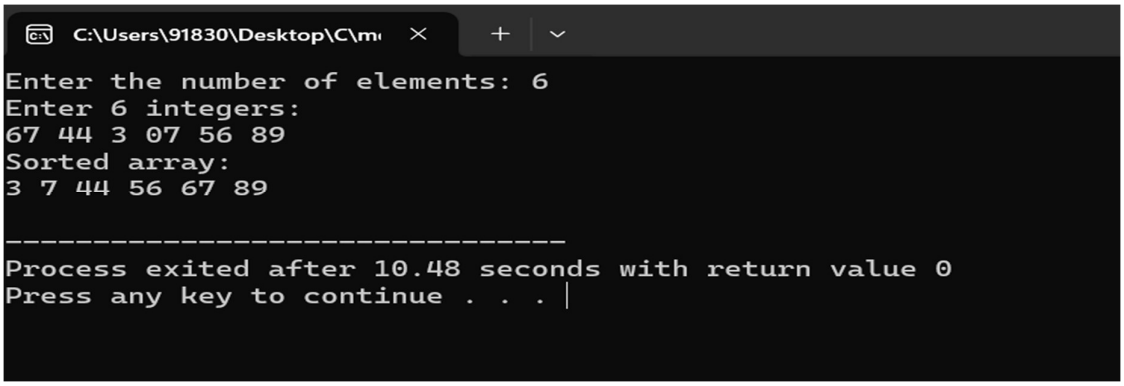
16) Write a program to perform Merge Sort.

```
#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) {
        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);
    }
}
int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    mergeSort(arr, 0, n - 1);
    printf("Sorted array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

Output

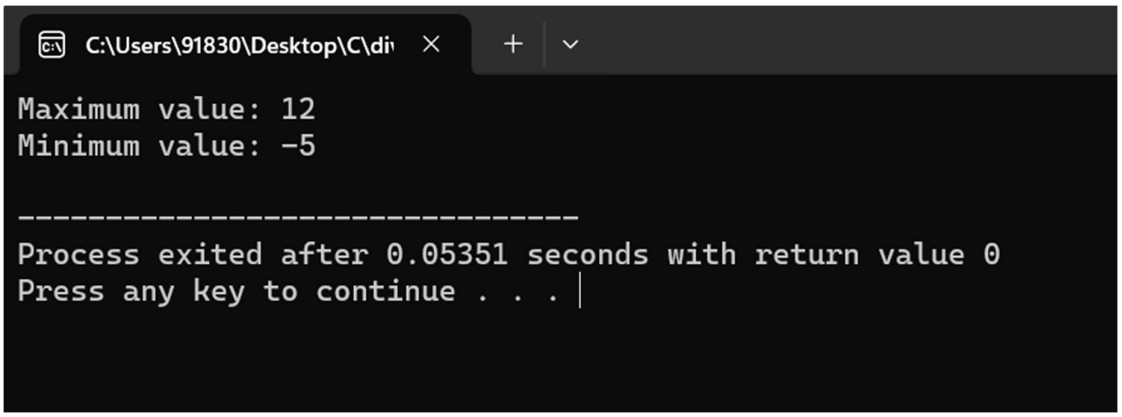


17) Using Divide and Conquer strategy to find Max and Min value in the list.

```
#include <stdio.h>
#include <limits.h>
void findMaxMin(int arr[], int low, int high, int *max, int *min) {
    if (low == high) {
        if (arr[low] > *max) *max = arr[low];
        if (arr[low] < *min) *min = arr[low];
        return;
    }
    if (high == low + 1) {
        if (arr[low] > arr[high]) {
            if (arr[low] > *max) *max = arr[low];
            if (arr[high] < *min) *min = arr[high];
        } else {
            if (arr[high] > *max) *max = arr[high];
            if (arr[low] < *min) *min = arr[low];
        }
        return;
    }
    int mid = (low + high) / 2;
    findMaxMin(arr, low, mid, max, min);
    findMaxMin(arr, mid + 1, high, max, min);
}
```

```
int main() {
    int arr[] = {3, 5, 1, 8, 7, -2, 0, 12, -5};
    int n = sizeof(arr) / sizeof(arr[0]);
    int max = INT_MIN;
    int min = INT_MAX;
    findMaxMin(arr, 0, n - 1, &max, &min);
    printf("Maximum value: %d\n", max);
    printf("Minimum value: %d\n", min);
    return 0;
}
```

Output



18) Write a program to generate all the prime numbers.

```
#include <stdio.h>
#include <stdlib.h>
void generatePrimes(int limit) {
    int *isPrime = malloc((limit + 1) * sizeof(int));
    for (int i = 0; i <= limit; i++) {
        isPrime[i] = 1;
    }
    isPrime[0] = isPrime[1] = 0;
    for (int p = 2; p * p <= limit; p++) {
        if (isPrime[p]) {
            for (int multiple = p * p; multiple <= limit; multiple += p) {
```

```
            isPrime[multiple] = 0;
        }
    }
}
printf("Prime numbers up to %d:\n", limit);
for (int i = 2; i <= limit; i++) {
    if (isPrime[i]) {
        printf("%d ", i);
    }
}
printf("\n");
free(isPrime);
}
int main() {
    int limit;
    printf("Enter the limit: ");
    scanf("%d", &limit);
    generatePrimes(limit);
    return 0;
}
```

Output

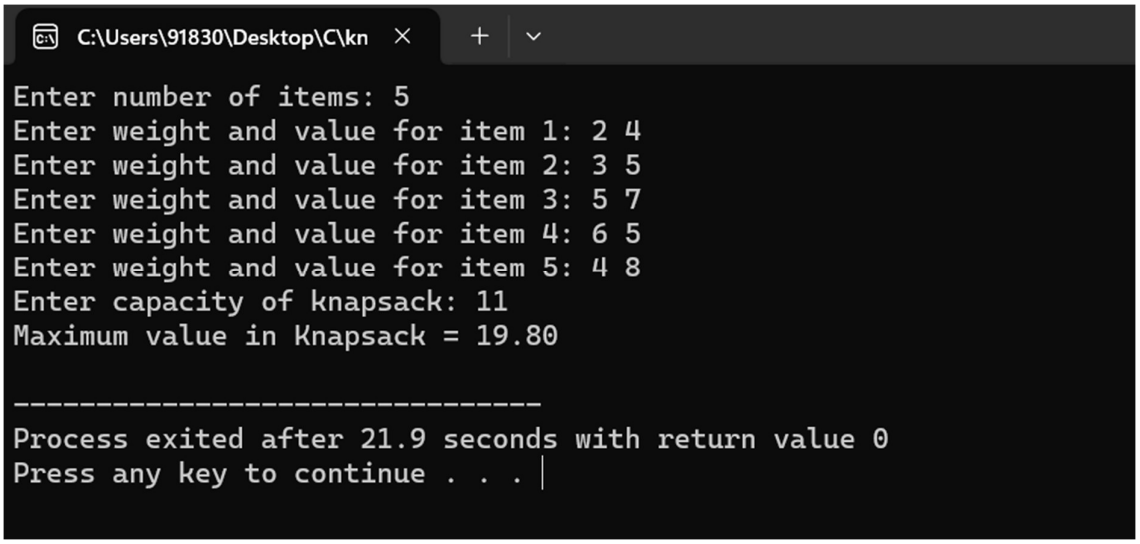
19) Write a program to perform Knapsack problem using greedy techniques.

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

```
typedef struct {
    int weight;
    int value;
    float ratio;
} Item;
void swap(Item *a, Item *b) {
    Item temp = *a;
    *a = *b;
    *b = temp;
}
void sortItems(Item items[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (items[j].ratio < items[j + 1].ratio) {
                swap(&items[j], &items[j + 1]);
            }
        }
    }
}
float knapsack(Item items[], int n, int capacity) {
    sortItems(items, n);
    float totalValue = 0.0;
    for (int i = 0; i < n; i++) {
        if (items[i].weight <= capacity) {
            capacity -= items[i].weight;
            totalValue += items[i].value;
        } else {
            totalValue += items[i].ratio * capacity;
            break;
        }
    }
    return totalValue;
```

```
}
int main() {
    int n, capacity;
    printf("Enter number of items: ");
    scanf("%d", &n);
    Item items[n];
    for (int i = 0; i < n; i++) {
        printf("Enter weight and value for item %d: ", i + 1);
        scanf("%d %d", &items[i].weight, &items[i].value);
        items[i].ratio = (float)items[i].value / items[i].weight;
    }
    printf("Enter capacity of knapsack: ");
    scanf("%d", &capacity);
    float maxValue = knapsack(items, n, capacity);
    printf("Maximum value in Knapsack = %.2f\n", maxValue);
    return 0;
}
```

Output



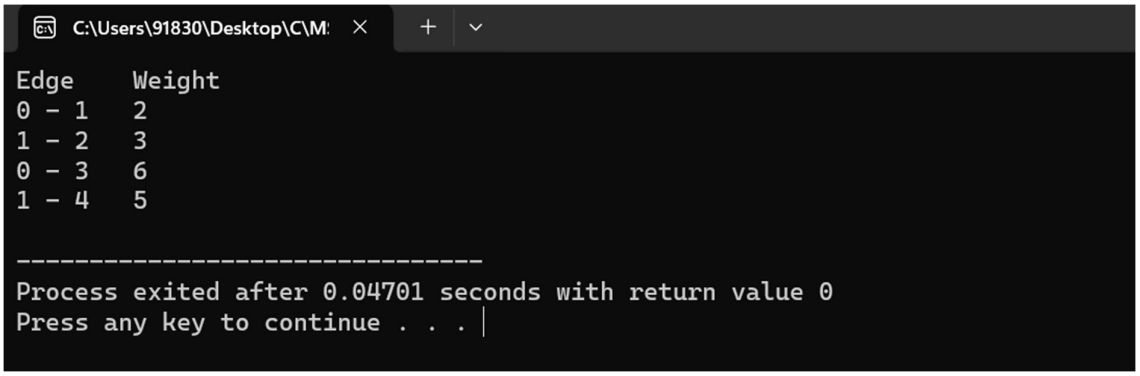
20) Write a program to perform MST using greedy techniques.

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

```
#define V 5
int minKey(int key[], int mstSet[]) {
    int min = INT_MAX, minIndex;
    for (int v = 0; v < V; v++) {
        if (mstSet[v] == 0 && key[v] < min) {
            min = key[v];
            minIndex = v;
        }
    }
    return minIndex;
}
void primMST(int graph[V][V]) {
    int parent[V];
    int key[V];
    int mstSet[V];
    for (int i = 0; i < V; i++) {
        key[i] = INT_MAX;
        mstSet[i] = 0;
    }
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = 1;
        for (int v = 0; v < V; v++) {
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }
```

```
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++) {
        printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
    }
}
int main() {
    int graph[V][V] = {
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0}
    };
    primMST(graph);
    return 0;
}
```

Output



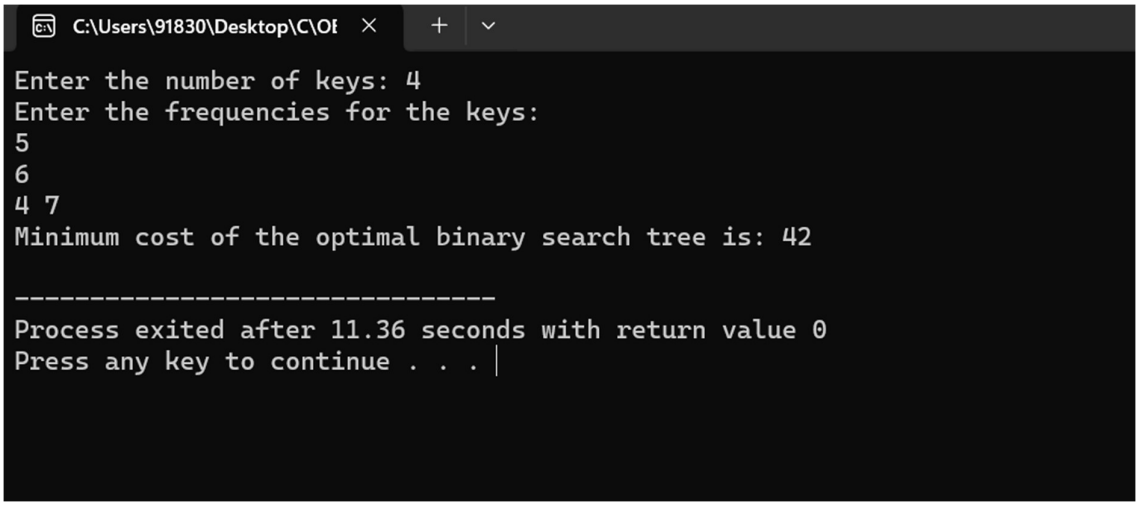
21) Using Dynamic programming concept to find out Optimal binary search tree.

```
#include <stdio.h>
#include <limits.h>
#define MAX 100
int cost[MAX][MAX];
int freq[MAX];
int n;
```

```
int sumFreq(int i, int j) {
    int sum = 0;
    for (int k = i; k <= j; k++) {
        sum += freq[k];
    }
    return sum;
}
void optimalBST() {
    for (int i = 0; i < n; i++) {
        cost[i][i] = freq[i];
    }
    for (int len = 2; len <= n; len++) {
        for (int i = 0; i <= n - len; i++) {
            int j = i + len - 1;
            cost[i][j] = INT_MAX;
            for (int r = i; r <= j; r++) {
                int c = ((r > i) ? cost[i][r - 1] : 0) +
                        ((r < j) ? cost[r + 1][j] : 0) +
                        sumFreq(i, j);
                if (c < cost[i][j]) {
                    cost[i][j] = c;
                }
            }
        }
    }
}
int main() {
    printf("Enter the number of keys: ");
    scanf("%d", &n);
    printf("Enter the frequencies for the keys:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &freq[i]);
```

```
    }
    optimalBST();
    printf("Minimum cost of the optimal binary search tree is: %d\n", cost[0][n - 1]);
    return 0;
}
```

Output

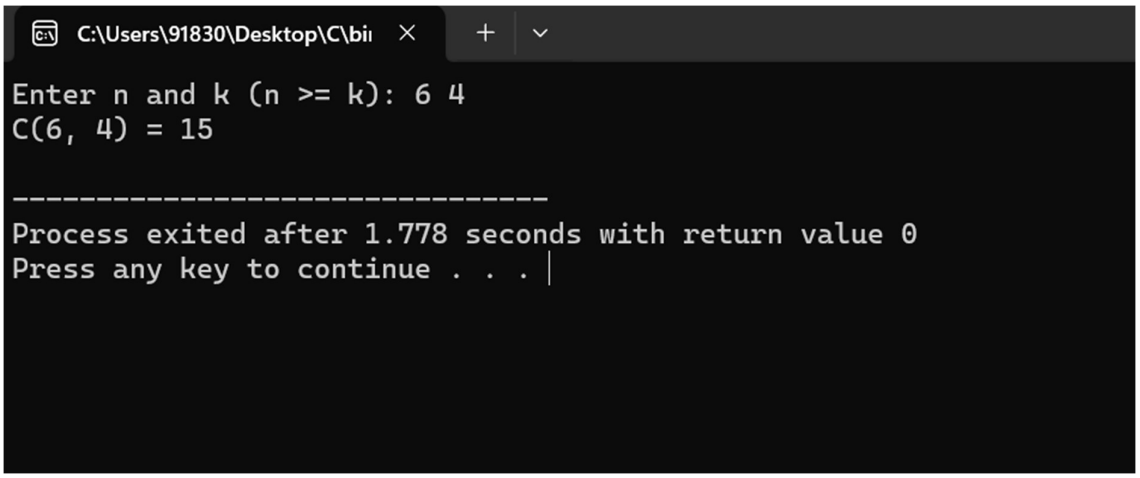


22) Using Dynamic programming techniques to find binomial coefficient of a given number

```
#include <stdio.h>
#define MAX 100
int binomialCoeff(int n, int k) {
    int C[MAX][MAX];
    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;
            } else {
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
            }
        }
    }
    return C[n][k];
}
```

```
int main() {
    int n, k;
    printf("Enter n and k (n >= k): ");
    scanf("%d %d", &n, &k);
    if (k > n) {
        printf("Invalid input: k cannot be greater than n.\n");
        return 1;
    }
    int result = binomialCoeff(n, k);
    printf("C(%d, %d) = %d\n", n, k, result);
    return 0;
}
```

Output

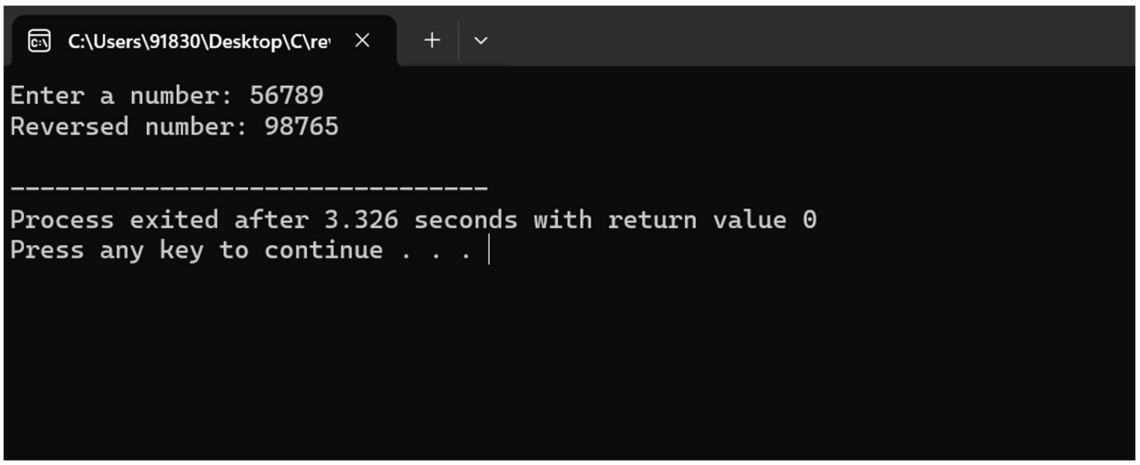


23) Write a program to find the reverse of a given number.

```
#include <stdio.h>
int reverseNumber(int num) {
    int reversed = 0;
    while (num != 0) {
        int digit = num % 10;
        reversed = reversed * 10 + digit;
        num /= 10;
    }
```

```
    return reversed;
}
int main() {
    int number;
    printf("Enter a number: ");
    scanf("%d", &number);
    int reversedNumber = reverseNumber(number);
    printf("Reversed number: %d\n", reversedNumber);
    return 0;
}
```

Output

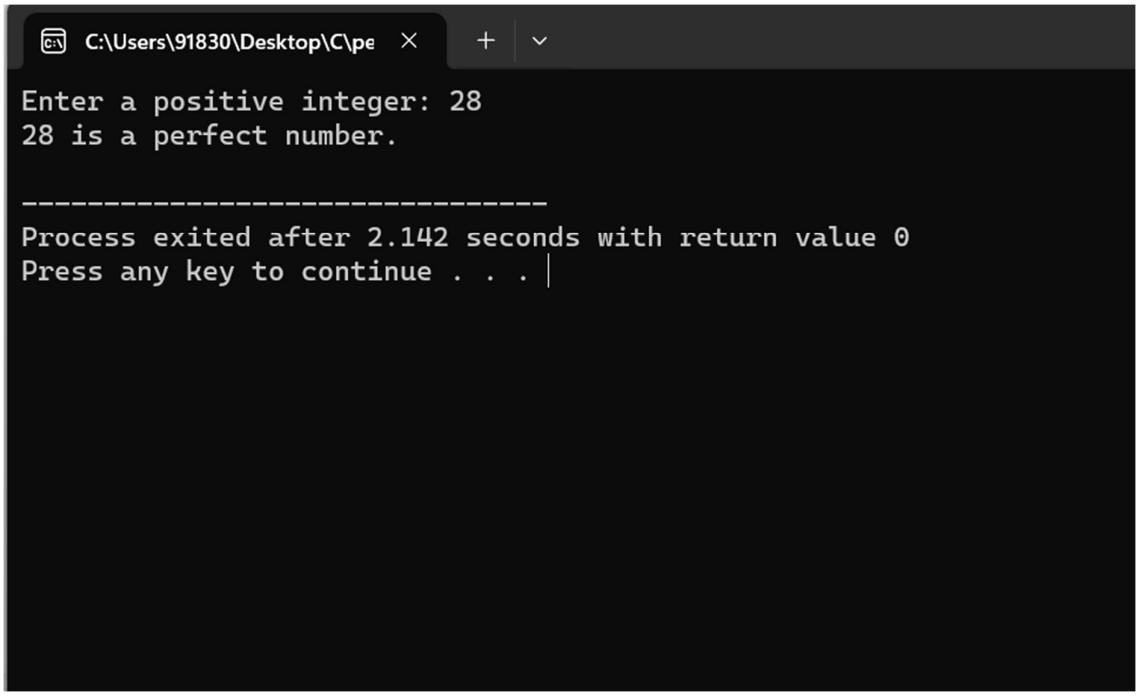


24) Write a program to find the perfect number.

```
#include <stdio.h>
int isPerfect(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 positive integer: ");
    scanf("%d", &number);
    if (number <= 0) {
        printf("Please enter a positive integer.\n");
        return 1;
    }
    if (isPerfect(number)) {
        printf("%d is a perfect number.\n", number);
    } else {
        printf("%d is not a perfect number.\n", number);
    }
    return 0;
}
```

Output

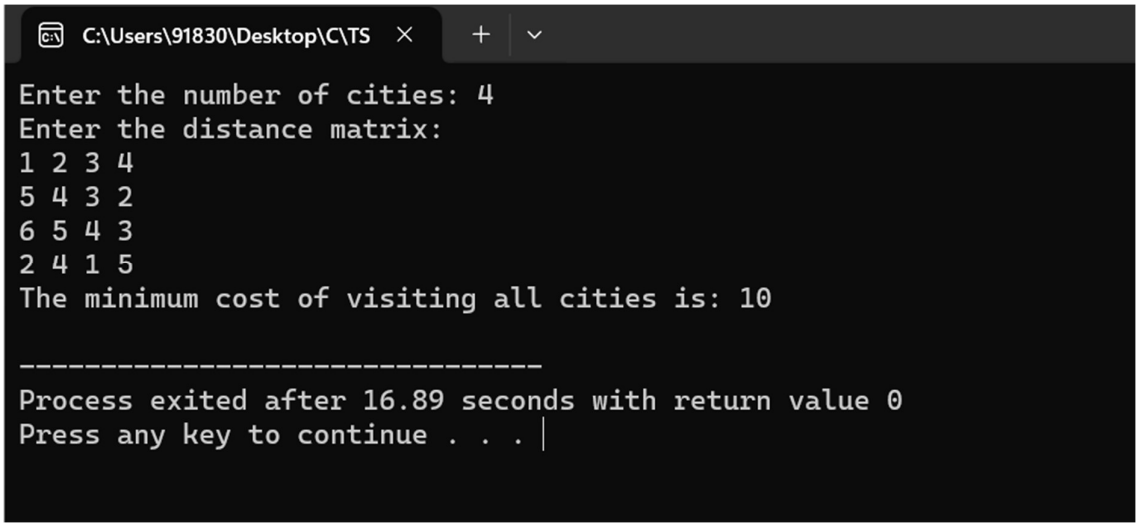


25) Write a program to perform travelling salesman problem using dynamic programming

```
#include <stdio.h>
#include <limits.h>
#define MAX 20
#define INF INT_MAX
int n;
int dist[MAX][MAX];
int dp[1 << MAX][MAX];
int visited_all;
int tsp(int mask, int pos) {
    if (mask == visited_all) {
        return dist[pos][0];
    }
    if (dp[mask][pos] != -1) {
        return dp[mask][pos];
    }
    int ans = INF;
    for (int city = 0; city < n; city++) {
        if ((mask & (1 << city)) == 0) {
            int newAns = dist[pos][city] + tsp(mask | (1 << city), city);
            ans = (ans < newAns) ? ans : newAns;
        }
    }
    return dp[mask][pos] = ans;
}
int main() {
    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 < (1 << n); i++) {
        for (int j = 0; j < n; j++) {
            dp[i][j] = -1;
        }
    }
    visited_all = (1 << n) - 1;
    int result = tsp(1, 0);
    printf("The minimum cost of visiting all cities is: %d\n", result);
    return 0;
}
```

Output



26) Write a program for the given pattern If n=4

1

1 2

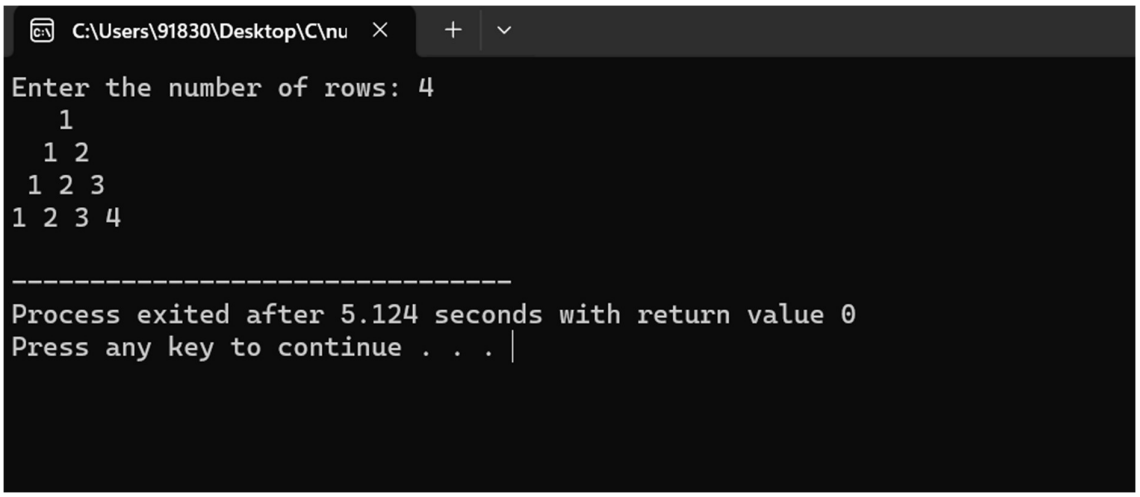
1 2 3

1 2 3 4

```
#include <stdio.h>
int main() {
    int rows, i, j;
    printf("Enter the number of rows: ");
```

```
    scanf("%d", &rows);
    for (i = 1; i <= rows; i++) {
        for (j = i; j < rows; j++) {
            printf(" ");
        }
        for (j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```

Output



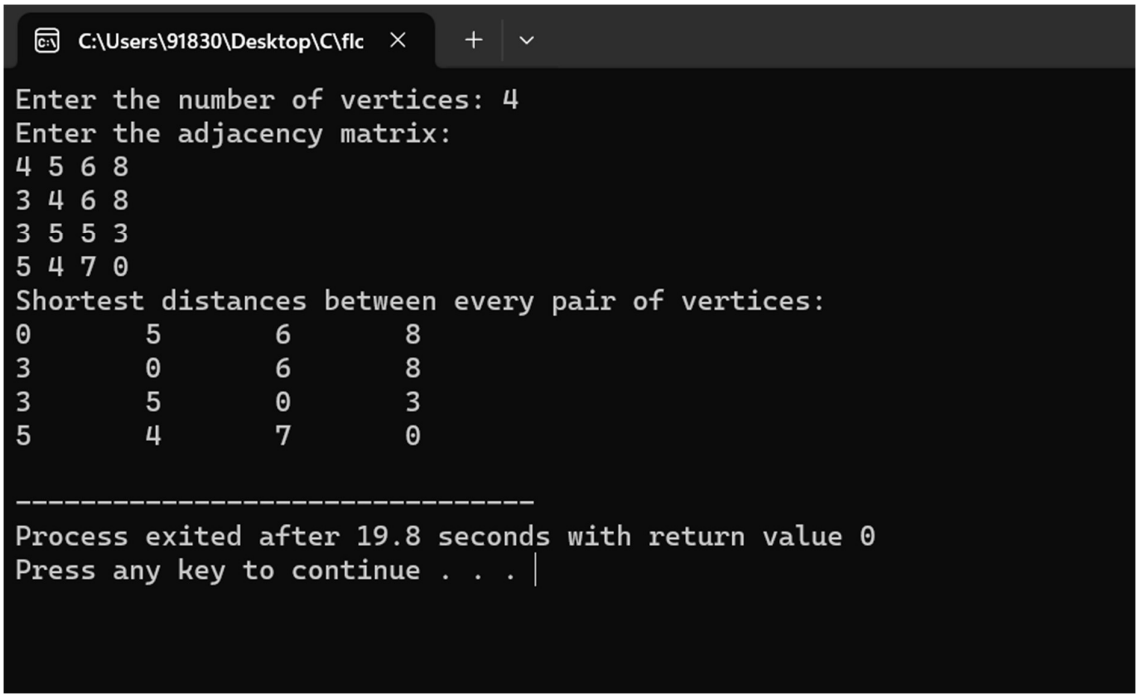
27) Write a program to perform Floyd's algorithm

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

```
if (i == j) {
dist[i][j] = 0;
} else if (graph[i][j] != 0) {
dist[i][j] = graph[i][j];
} else {
dist[i][j] = INF;
}
}
}
for (int k = 0; k < n; k++) {
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
if (dist[i][k] != INF && dist[k][j] != INF && dist[i][j] >
dist[i][k] + dist[k][j]) {
dist[i][j] = dist[i][k] + dist[k][j];
}
}
}
}
printf("Shortest distances between every pair of vertices:\n");
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
if (dist[i][j] == INF) {
printf("INF\t");
} else {
printf("%d\t", dist[i][j]);
}
}
printf("\n");
}
}
int main() {
```

```
int n;
printf("Enter the number of vertices: ");
scanf("%d", &n);
int graph[MAX][MAX];
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]);
if (i != j && graph[i][j] == 0) {
graph[i][j] = INF;
}
}
}
floydWarshall(graph, n);
return 0;
}
```

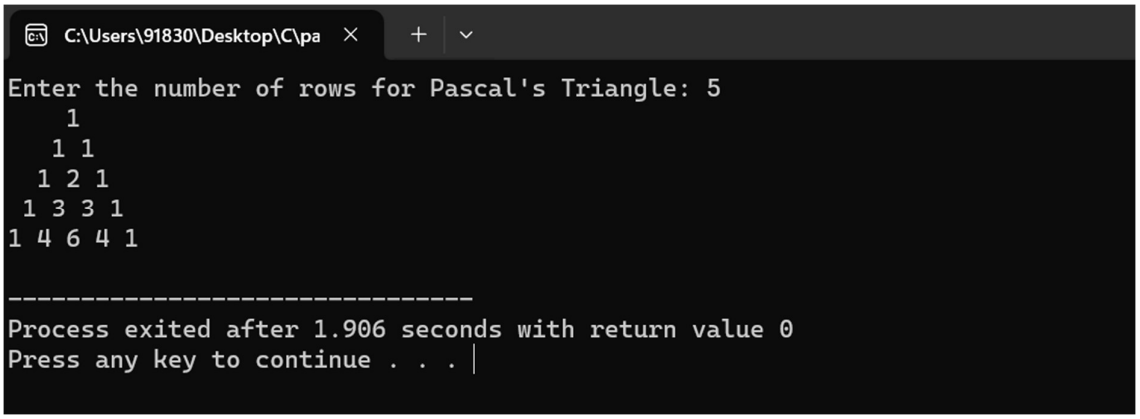
Output



28) Write a program for pascal triangle.

```
#include <stdio.h>
int main() {
    int rows, i, j, coefficient;
    printf("Enter the number of rows for Pascal's Triangle: ");
    scanf("%d", &rows);
    for (i = 0; i < rows; i++) {
        for (j = rows; j > i + 1; j--) {
            printf(" ");
        }
        coefficient = 1;
        for (j = 0; j <= i; j++) {
            printf("%d ", coefficient);
            coefficient = coefficient * (i - j) / (j + 1);
        }
        printf("\n");
    }
    return 0;
}
```

Output



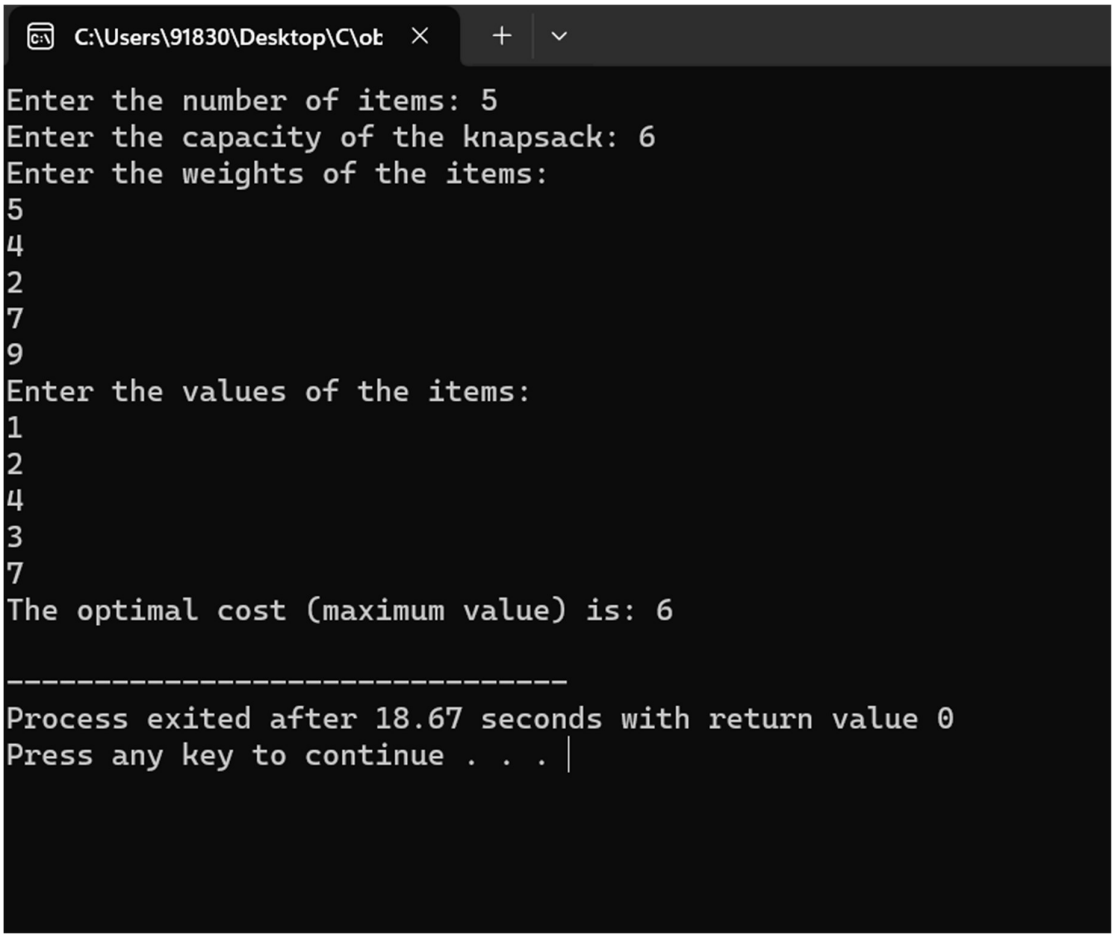
29) Write a program to find the optimal cost by using appropriate algorithm

```
#include <stdio.h>
int max(int a, int b) {
    return (a > b) ? a : b;
```

```
}
int knapsack(int capacity, int weights[], int values[], int n) {
    int i, w;
    int K[n + 1][capacity + 1];
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0) {
                K[i][w] = 0;
            } else if (weights[i - 1] <= w) {
                K[i][w] = max(values[i - 1] + K[i - 1][w - weights[i - 1]], K[i - 1][w]);
            } else {
                K[i][w] = K[i - 1][w];
            }
        }
    }
    return K[n][capacity];
}
int main() {
    int n, capacity;
    printf("Enter the number of items: ");
    scanf("%d", &n);
    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &capacity);
    int weights[n], values[n];
    printf("Enter the weights of the items:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &weights[i]);
    }
    printf("Enter the values of the items:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &values[i]);
    }
```

```
    int optimalCost = knapsack(capacity, weights, values, n);
    printf("The optimal cost (maximum value) is: %d\n", optimalCost);
    return 0;
}
```

Output



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

```
#include <stdio.h>
int main() {
    int number, sum = 0;
    printf("Enter an integer: ");
    scanf("%d", &number);
    if (number < 0) {
        number = -number;  // Make it positive
    }
```

```
    while (number > 0) {
        sum += number % 10;  // Add the last digit to the sum
        number /= 10;        // Remove the last digit
    }
    printf("The sum of the digits is: %d\n", sum);
    return 0;
}
```

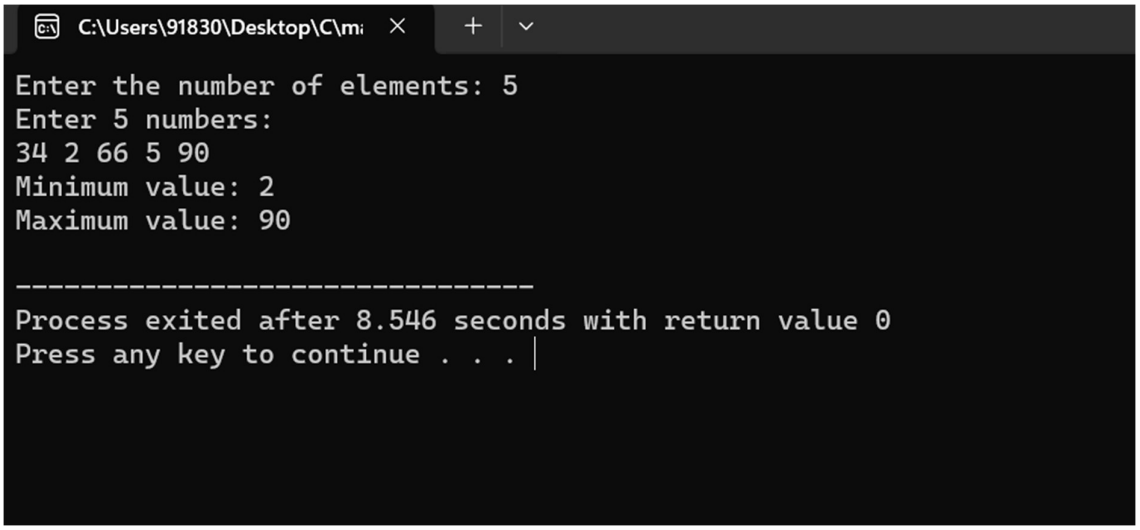
Output

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

```
#include <stdio.h>
int main() {
    int n, i, number;
    int min, max;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter %d numbers:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &number);
        if (i == 0) {
            min = max = number;
        } else {
            if (number < min) min = number;
            if (number > max) max = number;
        }
```

```
    }
    printf("Minimum value: %d\n", min);
    printf("Maximum value: %d\n", max);
    return 0;
}
```

Output



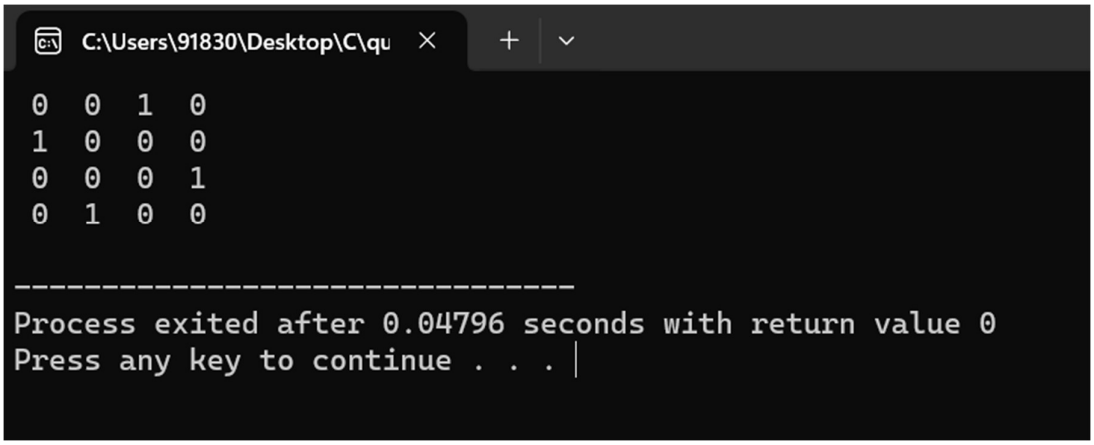
32) Write a program to perform n Queens problem using backtracking.

```
#include <stdio.h>
#include <stdbool.h>
#define N 4
void printSolution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf(" %d ", board[i][j]);
        }
        printf("\n");
    }
}

bool isSafe(int board[N][N], int row, int col) {
    for (int i = 0; i < col; i++) {
```

```
        if (board[row][i]) return false;
    }
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j]) return false;
    }
    for (int i = row, j = col; j >= 0 && i < N; i++, j--) {
        if (board[i][j]) return false;
    }
    return true;
}
bool solveNQUtil(int board[N][N], int col) {
    if (col >= N) return true;
    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            if (solveNQUtil(board, col + 1)) return true;
            board[i][col] = 0;
        }
    }
    return false;
}
void solveNQ() {
    int board[N][N] = {0};
    if (!solveNQUtil(board, 0)) {
        printf("Solution does not exist\n");
    } else {
        printSolution(board);
    }
}
int main() {
    solveNQ();
    return 0;
```

}

Output

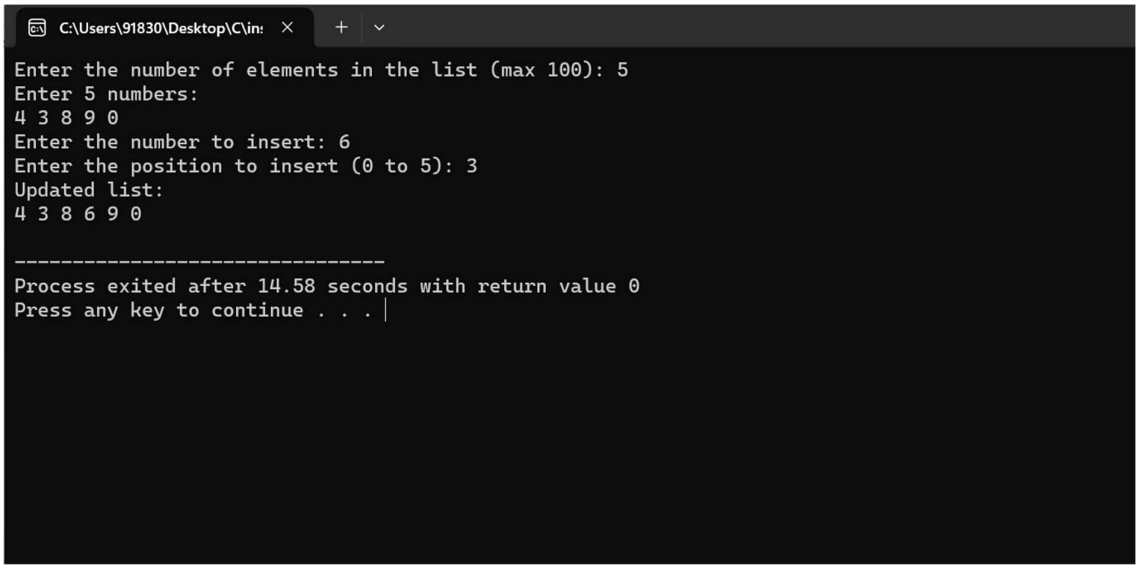


33) Write a program to insert a number in a list.

```
#include <stdio.h>
#define MAX_SIZE 100
int main() {
    int list[MAX_SIZE];
    int n, i, number, position;
    printf("Enter the number of elements in the list (max %d): ", MAX_SIZE);
    scanf("%d", &n);
    printf("Enter %d numbers:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &list[i]);
    }
    printf("Enter the number to insert: ");
    scanf("%d", &number);
    printf("Enter the position to insert (0 to %d): ", n);
    scanf("%d", &position);
    if (position < 0 || position > n) {
        printf("Invalid position!\n");
        return 1;
    }
    for (i = n; i > position; i--) {
```

```
        list[i] = list[i - 1];
    }
    list[position] = number;
    n++;
    printf("Updated list:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", list[i]);
    }
    printf("\n");
    return 0;
}
```

Output

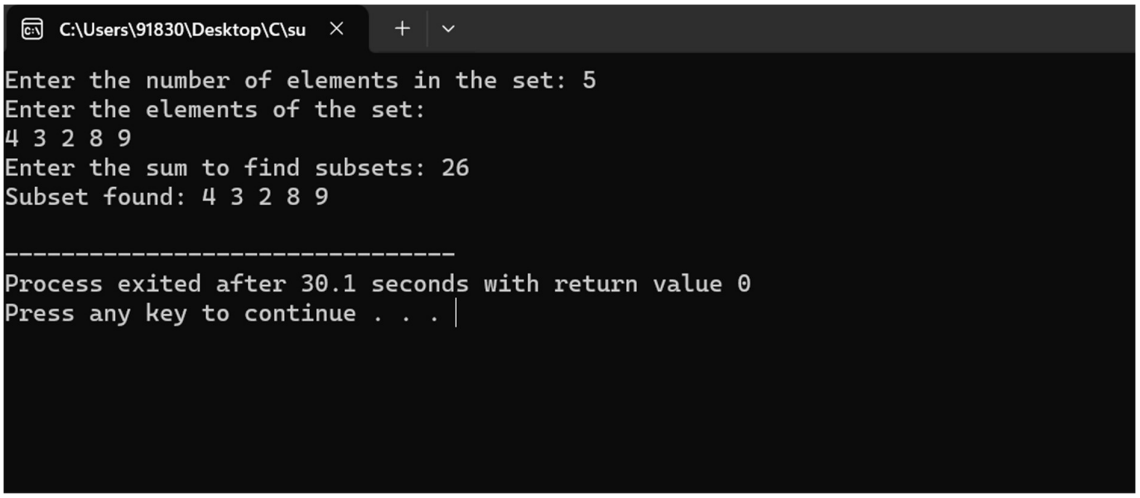


34) Write a program to perform sum of subsets problem using backtracking

```
#include <stdio.h>
void findSubset(int set[], int n, int sum, int currentSum, int subset[], int index) {
    if (currentSum == sum) {
        printf("Subset found: ");
        for (int i = 0; i < index; i++) {
            printf("%d ", subset[i]);
        }
```

```
        printf("\n");
        return;
    }
    if (currentSum > sum || n == 0) {
        return;
    }
    subset[index] = set[0];
    findSubset(set + 1, n - 1, sum, currentSum + set[0], subset, index + 1);
    findSubset(set + 1, n - 1, sum, currentSum, subset, index);
}
int main() {
    int n, sum;
    printf("Enter the number of elements in the set: ");
    scanf("%d", &n);
    int set[n];
    printf("Enter the elements of the set:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &set[i]);
    }
    printf("Enter the sum to find subsets: ");
    scanf("%d", &sum);
    int subset[n];
    findSubset(set, n, sum, 0, subset, 0);
    return 0;
}
```

Output

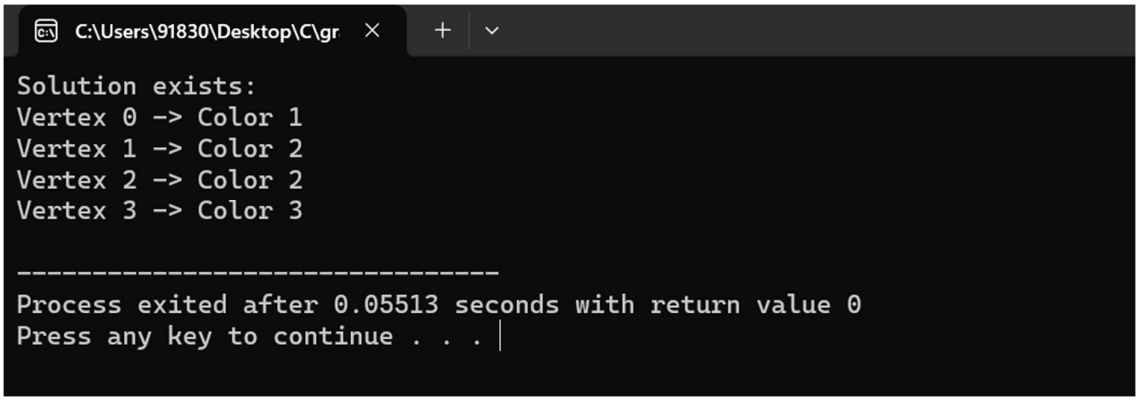


35) Write a program to perform graph coloring problem using backtracking.

```
#include <stdio.h>
#include <stdbool.h>
#define V 4
bool isSafe(int graph[V][V], int color[], int v, int c) {
    for (int i = 0; i < V; i++) {
        if (graph[v][i] && color[i] == c) {
            return false;
        }
    }
    return true;
}
bool graphColoringUtil(int graph[V][V], int m, int color[], int v) {
    if (v == V) {
        return true;
    }
    for (int c = 1; c <= m; c++) {
        if (isSafe(graph, color, v, c)) {
            color[v] = c;
            if (graphColoringUtil(graph, m, color, v + 1)) {
                return true;
            }
```

```
            color[v] = 0; // Backtrack
        }
    }
    return false;
}
void graphColoring(int graph[V][V], int m) {
    int color[V];
    for (int i = 0; i < V; i++) {
        color[i] = 0;
    }
    if (graphColoringUtil(graph, m, color, 0)) {
        printf("Solution exists: \n");
        for (int i = 0; i < V; i++) {
            printf("Vertex %d -> Color %d\n", i, color[i]);
        }
    } else {
        printf("No solution exists.\n");
    }
}
int main() {
    int graph[V][V] = {
        {0, 1, 1, 1},
        {1, 0, 0, 1},
        {1, 0, 0, 1},
        {1, 1, 1, 0}
    };
    int m = 3;
    graphColoring(graph, m);
    return 0;
}
```

Output

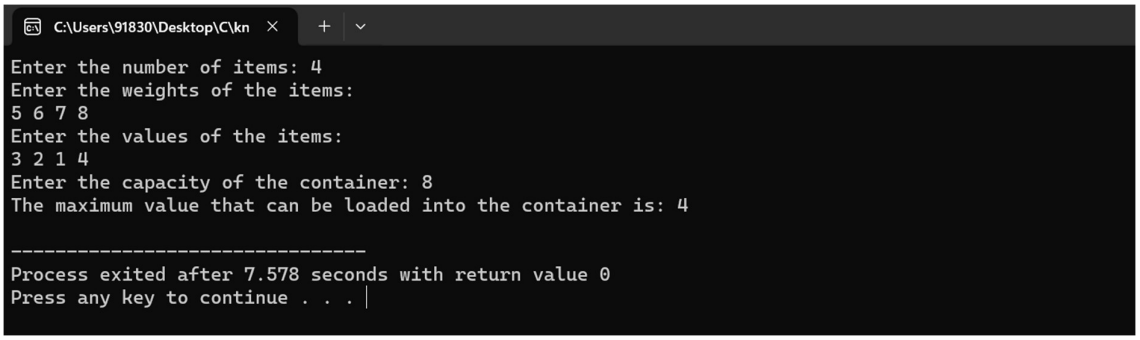


36) Write a program to compute container loader Problem.

```
#include <stdio.h>
#define MAX_CAPACITY 100
int max(int a, int b) {
    return (a > b) ? a : b;
}
int knapsack(int weights[], int values[], int n, int capacity) {
    int dp[n + 1][capacity + 1];
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0) {
                dp[i][w] = 0;
            } else if (weights[i - 1] <= w) {
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }
    return dp[n][capacity];
}
int main() {
    int n, capacity;
    printf("Enter the number of items: ");
```

```
    scanf("%d", &n);
    int weights[n], values[n];
    printf("Enter the weights of the items:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &weights[i]);
    }
    printf("Enter the values of the items:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &values[i]);
    }
    printf("Enter the capacity of the container: ");
    scanf("%d", &capacity);
    int maxValue = knapsack(weights, values, n, capacity);
    printf("The maximum value that can be loaded into the container is: %d\n", maxValue);
    return 0;
}
```

Output

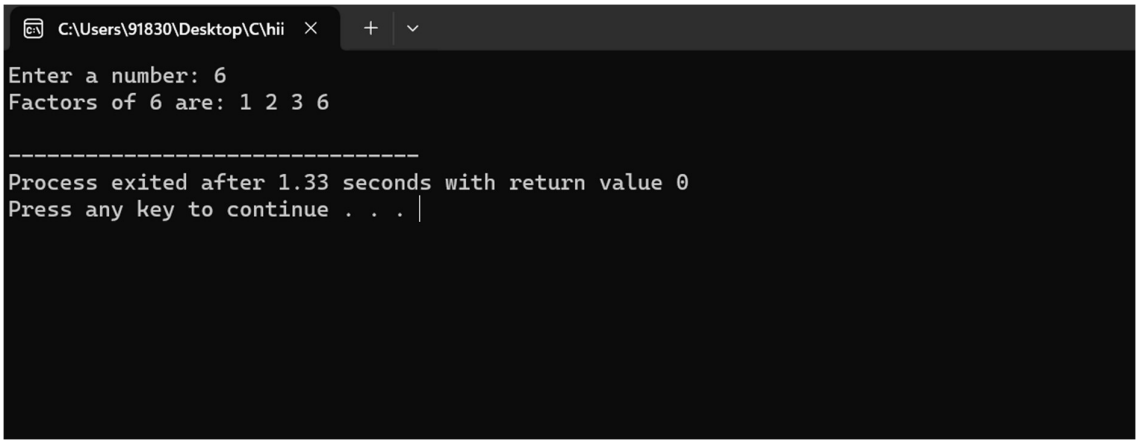


37) Write a program to generate the list of all factor for n value.

```
#include <stdio.h>
int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    printf("Factors of %d are: ", 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

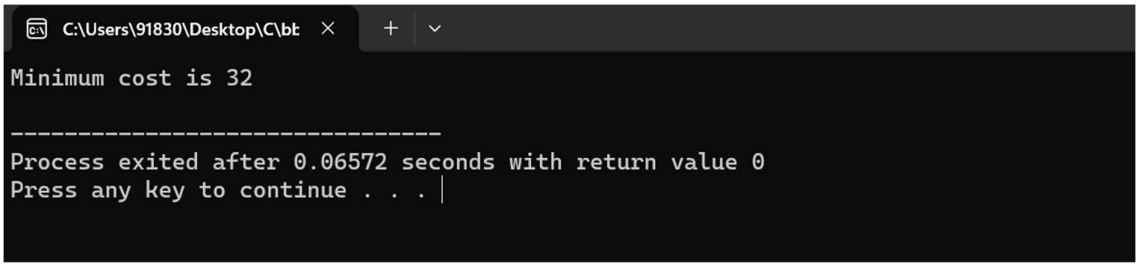
```
#include <stdio.h>
#include <limits.h>
#define N 4 // Number of tasks and workers
void assignmentProblem(int costMatrix[N][N]);
int branchAndBound(int costMatrix[N][N], int assignment[], int row, int n,
int bound, int currCost, int minCost, int visited[]);
int calculateLowerBound(int costMatrix[N][N], int assignment[], int n, int
row, int visited[]);
int findMinCost(int costMatrix[N][N], int assignment[], int n, int
currCost, int minCost, int visited[]);
int main() {
int costMatrix[N][N] = {
{10, 2, 8, 12},
{9, 4, 7, 6},
```

```
{5, 11, 13, 10},
{7, 9, 16, 5}
};
assignmentProblem(costMatrix);
return 0;
}
void assignmentProblem(int costMatrix[N][N]) {
int assignment[N] = {-1}; // Store the assignment of tasks to workers
int visited[N] = {0}; // Track visited nodes
int minCost = INT_MAX; // Initialize minimum cost to a large value
minCost = branchAndBound(costMatrix, assignment, 0, N, 0, 0, minCost,
visited);
printf("Minimum cost is %d\n", minCost);
}
int branchAndBound(int costMatrix[N][N], int assignment[], int row, int n,
int bound, int currCost, int minCost, int visited[]) {
if (row == n) {
if (currCost < minCost) {
minCost = currCost;
}
return minCost;
}
for (int col = 0; col < n; col++) {
if (!visited[col]) {
visited[col] = 1;
assignment[row] = col;

int newBound = bound + costMatrix[row][col];
int lowerBound = calculateLowerBound(costMatrix, assignment, n,
row + 1, visited);
if (newBound + lowerBound < minCost) {
minCost = branchAndBound(costMatrix, assignment, row + 1,
```

```
n, newBound, currCost + costMatrix[row][col], minCost, visited);
}
visited[col] = 0;
assignment[row] = -1;
}
}
return minCost;
}
int calculateLowerBound(int costMatrix[N][N], int assignment[], int n, int
row, int visited[]) {
int bound = 0;
for (int i = row; i < n; i++) {
int min1 = INT_MAX, min2 = INT_MAX;
for (int j = 0; j < n; j++) {
if (!visited[j] && costMatrix[i][j] < min1) {
min2 = min1;
min1 = costMatrix[i][j];
} else if (!visited[j] && costMatrix[i][j] < min2) {
min2 = costMatrix[i][j];
}
}
bound += (min1 == INT_MAX) ? 0 : min1;
bound += (min2 == INT_MAX) ? 0 : min2;
}
for (int j = 0; j < n; j++) {
int min1 = INT_MAX, min2 = INT_MAX;
for (int i = row; i < n; i++) {
if (!visited[j] && costMatrix[i][j] < min1) {
min2 = min1;
min1 = costMatrix[i][j];
} else if (!visited[j] && costMatrix[i][j] < min2) {
min2 = costMatrix[i][j];
```

```
}
}
bound += (min1 == INT_MAX) ? 0 : min1;
bound += (min2 == INT_MAX) ? 0 : min2;
}
return bound / 2;
}
```

Output

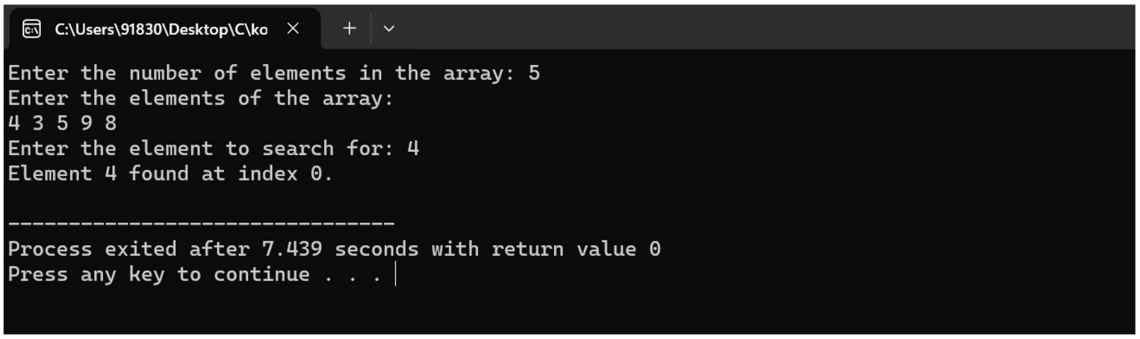


39) Write a program for to perform liner search.

```
#include <stdio.h>
int linearSearch(int arr[], int size, int target) {
for (int i = 0; i < size; i++) {
if (arr[i] == target) {
return i;
}
}
return -1;
}
int main() {
int arr[100];
int size, target, result;
printf("Enter the number of elements in the array: ");
scanf("%d", &size);
printf("Enter the elements of the array:\n");
for (int i = 0; i < size; i++) {
scanf("%d", &arr[i]);
```

```
}
printf("Enter the element to search for: ");
scanf("%d", &target);
result = linearSearch(arr, size, 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;
}
```

Output



40) Write a program to find out Hamiltonian circuit Using backtracking method

```
#include <stdio.h>
#include <stdbool.h>
#define V 5
bool isSafe(int graph[V][V], int path[], int pos, int v) {
    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 hamCycleUtil(int graph[V][V], int path[], int pos) {
    if (pos == V) {
        return (graph[path[pos - 1]][path[0]] == 1);
    }
    for (int v = 1; v < V; v++) {
        if (isSafe(graph, path, pos, v)) {
            path[pos] = v;
            if (hamCycleUtil(graph, path, pos + 1)) {
                return true;
            }
            path[pos] = -1; // Backtrack
        }
    }
    return false;
}
void hamCycle(int graph[V][V]) {
    int path[V];
    for (int i = 0; i < V; i++) {
        path[i] = -1;
    }
    path[0] = 0; // Start from vertex 0
    if (!hamCycleUtil(graph, path, 1)) {
        printf("Solution does not exist\n");
    } else {
        printf("Hamiltonian Circuit: ");
        for (int i = 0; i < V; i++) {
            printf("%d ", path[i]);
        }
        printf("%d\n", path[0]);
    }
```

```
}
int main() {
    int graph[V][V] = {
        {0, 1, 1, 1, 0},
        {1, 0, 0, 1, 1},
        {1, 0, 0, 0, 1},
        {1, 1, 0, 0, 1},
        {0, 1, 1, 1, 0}
    };
    hamCycle(graph);
    return 0;
}
```

Output