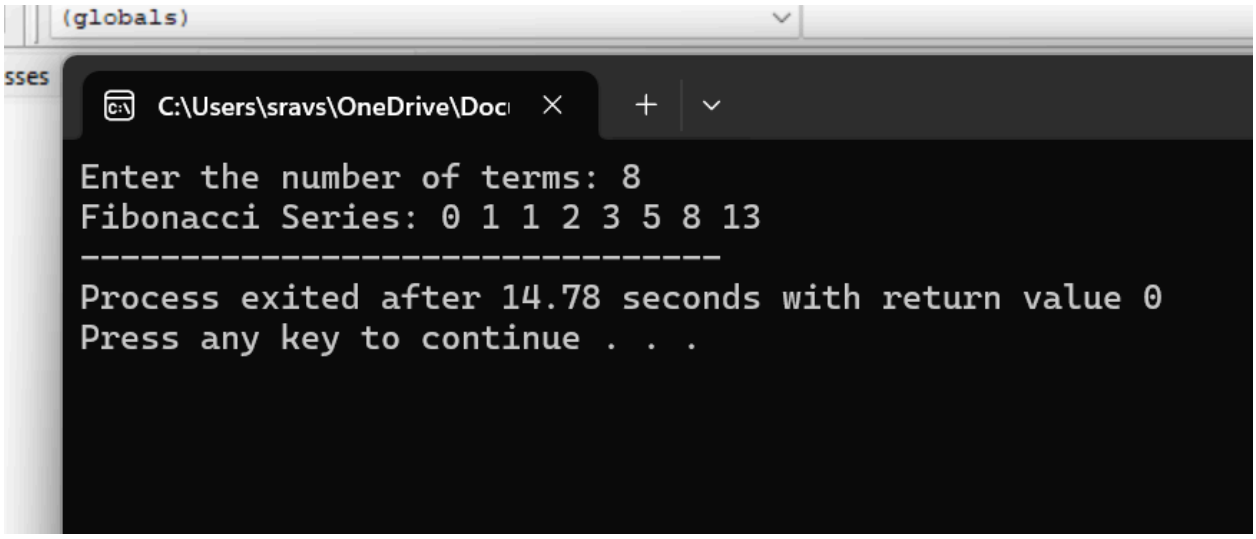
DAA LAB - DAY 01

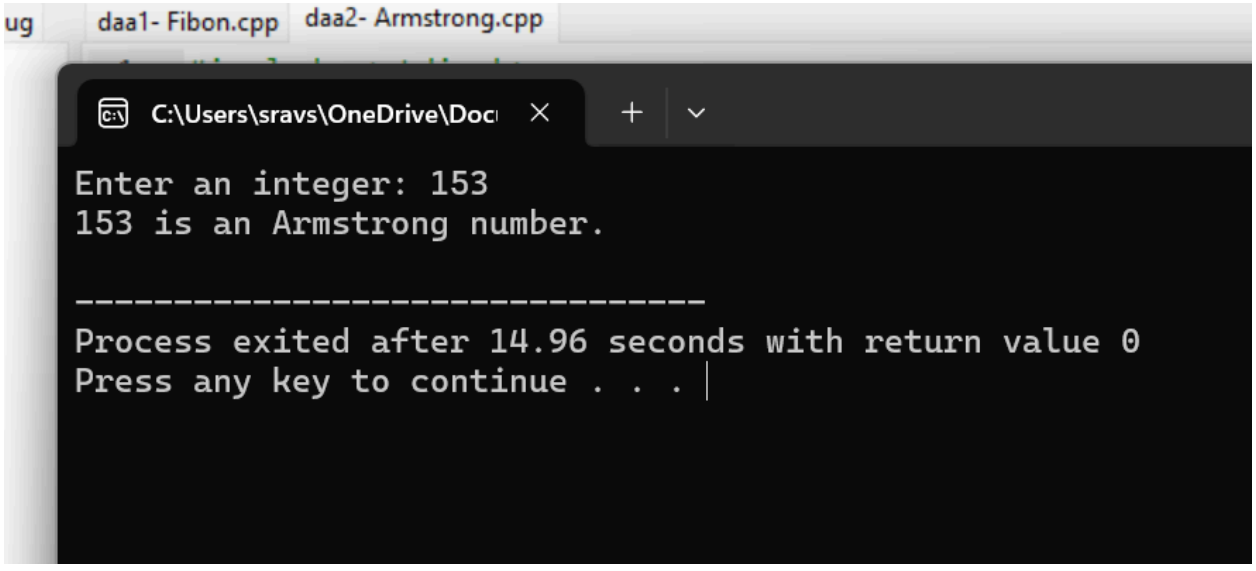
1. Fibonacci Series using recursion.

```
#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;
}
```



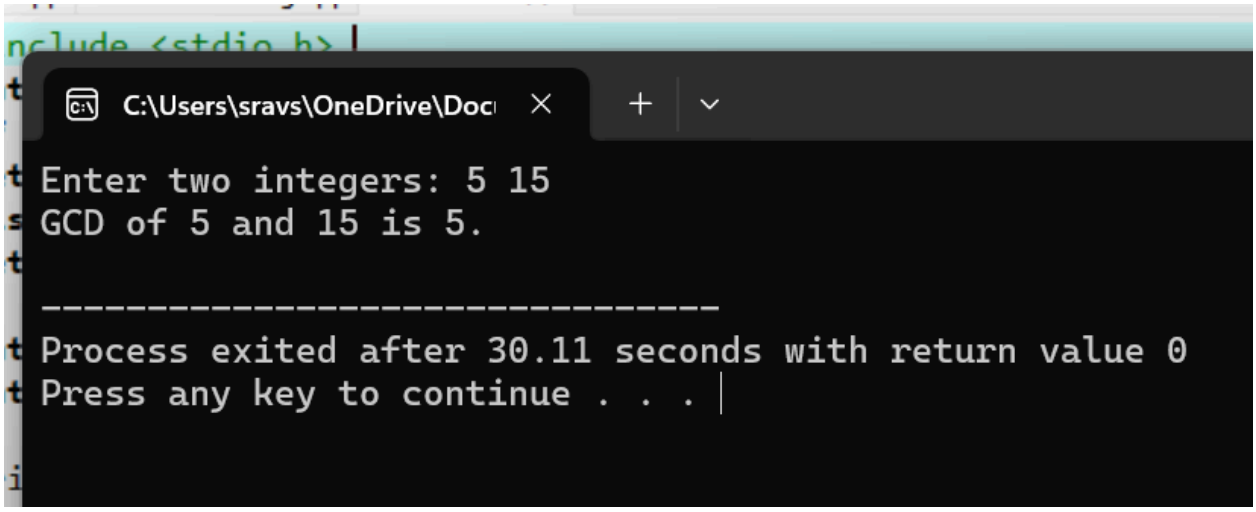
2.Armstrong or not.

```
#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;
}
```



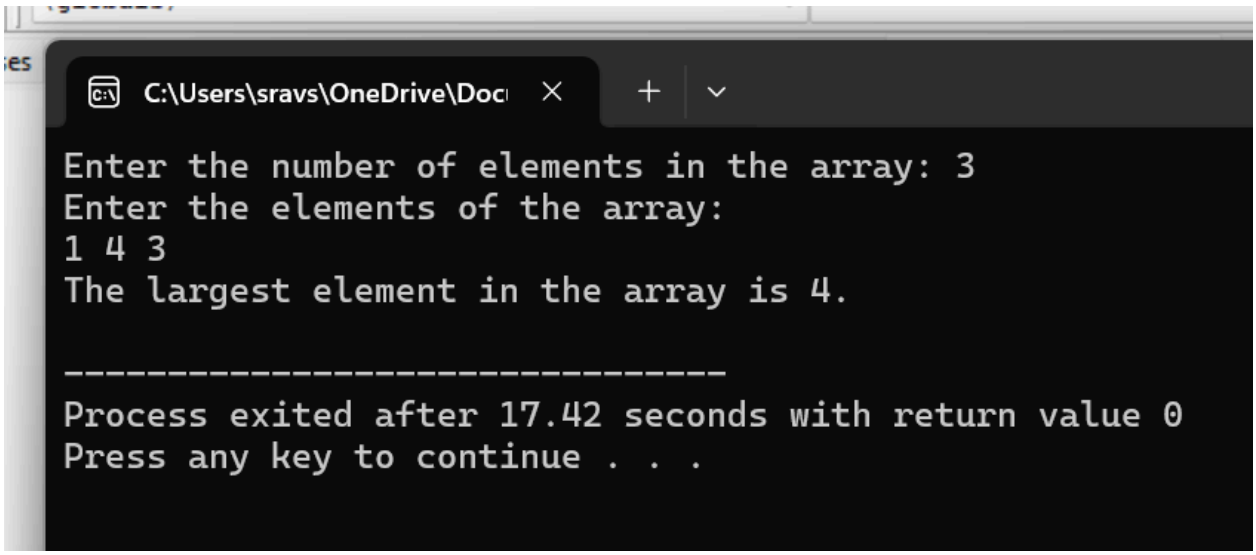
3.GCD of any two numbers

```
#include <stdio.h>
int gcd(int a, int b) {
if (b == 0)
return a;
else
return gcd(b, a % b);
}
int main() {
int num1, num2, result;

printf("Enter two integers: ");
scanf("%d %d", &num1, &num2);

result = gcd(num1, num2);

printf("GCD of %d and %d is %d.\n", num1, num2, result);
return 0;
}
```



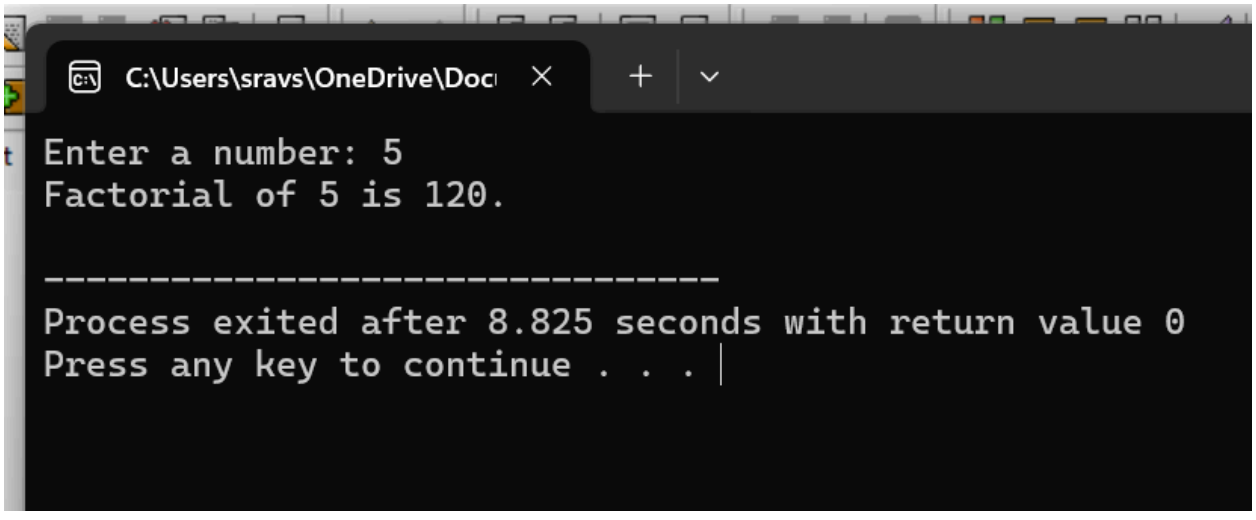
4. Largest element of an array

```
#include <stdio.h>
int main() {
int n, i;
int largest;

printf("Enter the number of elements in the array: ");
scanf("%d", &n);
int arr[n];

printf("Enter the elements of the array:\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 in the array is %d.\n", largest);
return 0;
}
```



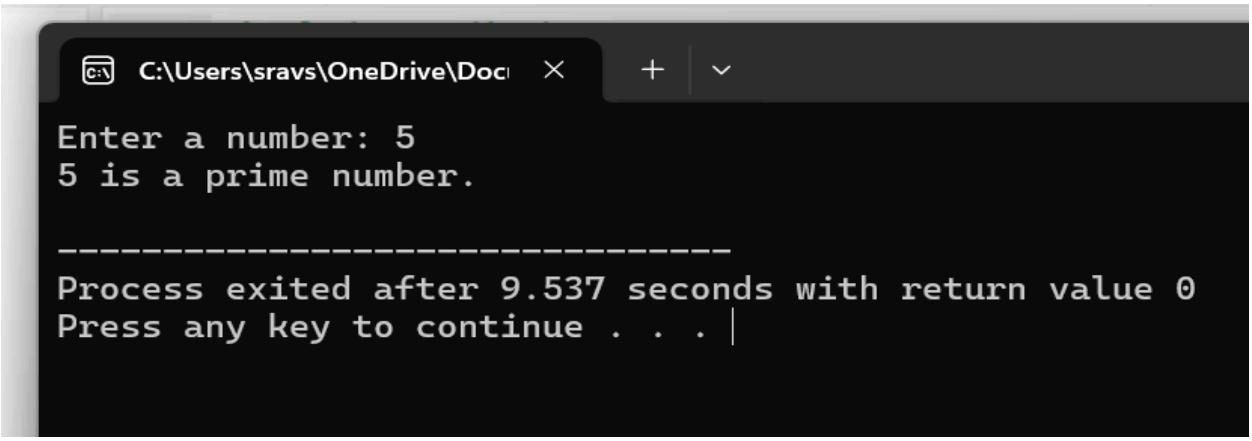
5. Factorial of a number

```
#include <stdio.h>
unsigned long long factorial(int n) {
unsigned long long fact = 1;
for (int i = 1; i <= n; ++i) {
fact *= i;
}
return fact;
}
int main() {
int num;

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

if (num < 0) {
printf("Factorial is not defined for negative numbers.\n");
} else {

printf("Factorial of %d is %llu.\n", num, factorial(num));
}
return 0;
}
```



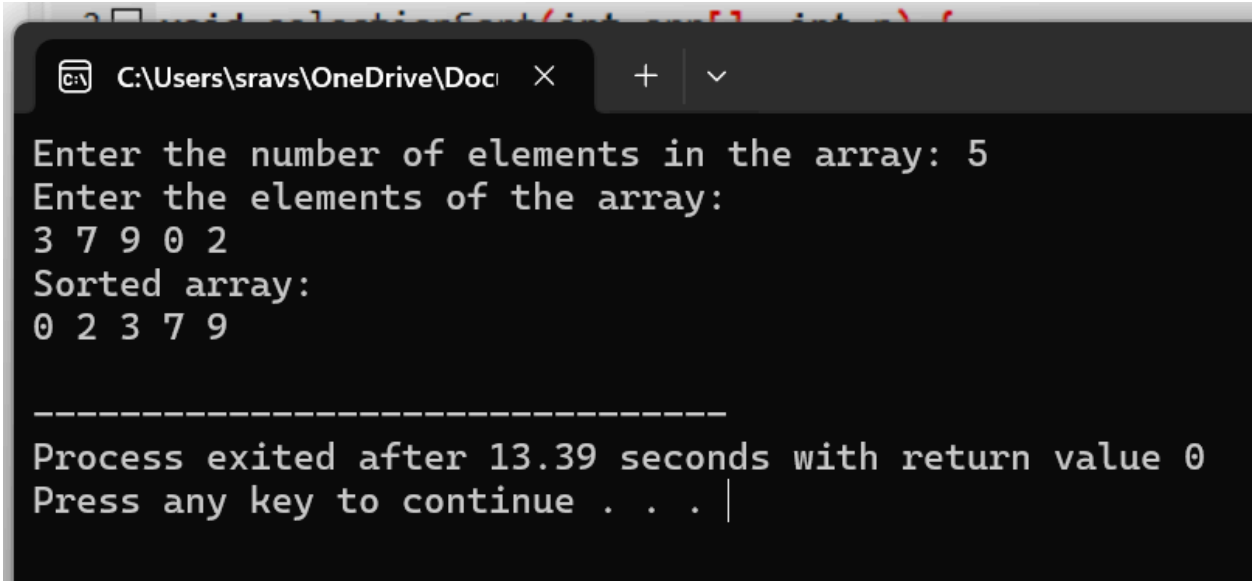
6. Number is Prime or Not.

```
#include <stdio.h>
#include <stdbool.h>
#include <math.h>
bool isPrime(int num) {
if (num <= 1) {
return false;
}
for (int i = 2; i <= sqrt(num); i++) {
if (num % i == 0) {
return false;
}
}
return true;
}
int main() {
int num;
printf("Enter a number: ");
scanf("%d", &num);
if (isPrime(num)) {
printf("%d is a prime number.\n", num);
} else {
printf("%d is not a prime number.\n", num);
}
return 0;
}
```



7. Selection sort

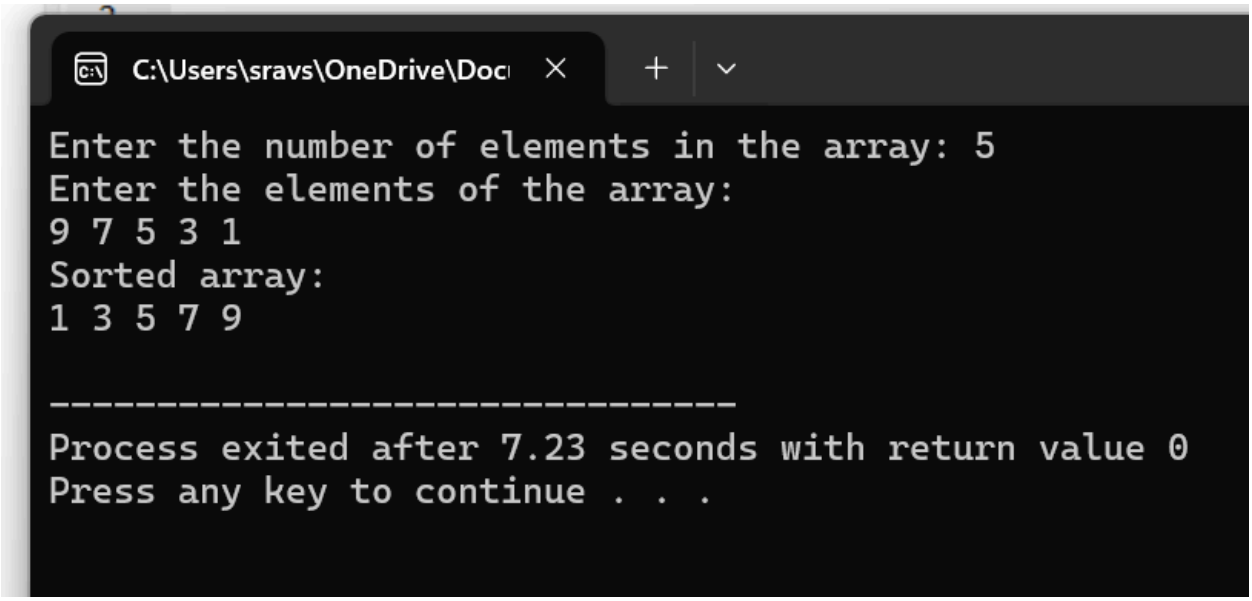
```
#include <stdio.h>
void selectionSort(int arr[], int n) {
int i, j, min_idx, temp;
   for (i = 0; i < n-1; i++) {
      min_idx = i;
      for (j = i+1; j < n; j++) {
         if (arr[j] < arr[min_idx]) {
            min_idx = j;
         }
      }
      temp = arr[min_idx];
      arr[min_idx] = arr[i];
      arr[i] = temp;
   }
}
void printArray(int arr[], int size) {
   int i;
   for (i = 0; i < size; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
}

int main() {
   int n, i;
   printf("Enter the number of elements in the array: ");
   scanf("%d", &n);
   int arr[n];
   printf("Enter the elements of the array:\n");
   for (i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }
   selectionSort(arr, n);
   printf("Sorted array: \n");
   printArray(arr, n);

   return 0;
}
```



8. 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;
}
}
}
}
void printArray(int arr[], int size) {
int i;
for (i = 0; i < size; i++) {
printf("%d ", arr[i]);
}
printf("\n");
```

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

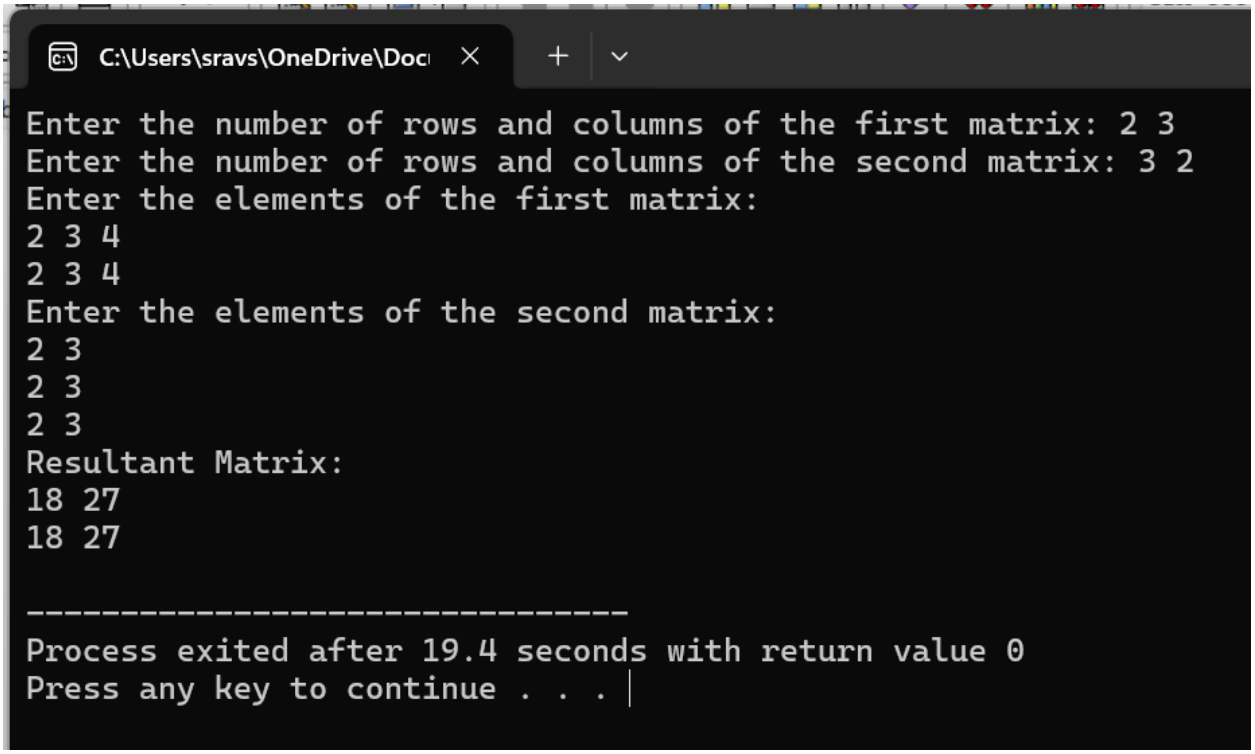


DAY- 02

9. Matrix Multiplication

```
#include <stdio.h>
void multiplyMatrices(int firstMatrix[][10], int secondMatrix[][10], int
resultMatrix[][10], int r1, int c1, int r2, int c2) {
    int i, j, k;
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c2; j++) {
            resultMatrix[i][j] = 0;
        }
    }
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c2; j++) {
            for (k = 0; k < c1; k++) {
                resultMatrix[i][j] += firstMatrix[i][k] *
secondMatrix[k][j];
            }
        }
    }
}
void printMatrix(int matrix[][10], int row, int col) {
    int i, j;
    for (i = 0; i < row; i++) {
        for (j = 0; j < col; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int r1, c1, r2, c2, i, j;
    printf("Enter the number of rows and columns of the first matrix: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter the number of rows and columns of the second matrix: ");
    scanf("%d %d", &r2, &c2);
    if (c1 != r2) {
```

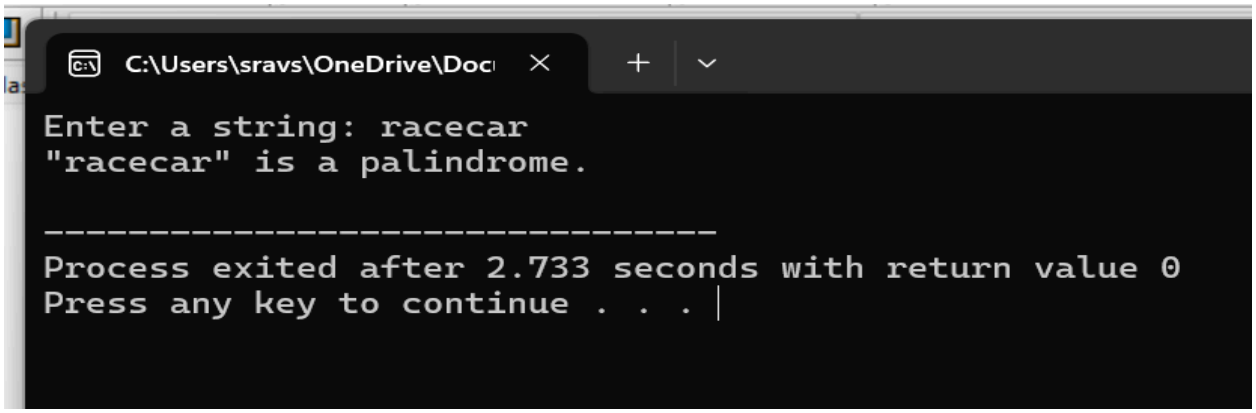
```
      printf("Error! Column of the first matrix must be equal to row of the second
matrix.\n");
      return -1;
   }
   int firstMatrix[10][10], secondMatrix[10][10], resultMatrix[10][10];

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

printf("Enter the elements of the second matrix:\n");
for (i = 0; i < r2; i++) {
for (j = 0; j < c2; j++) {
scanf("%d", &secondMatrix[i][j]);
}
}
multiplyMatrices(firstMatrix, secondMatrix, resultMatrix, r1, c1, r2,
c2);

printf("Resultant Matrix:\n");
printMatrix(resultMatrix, r1, c2);
return 0;
}
```



10. Palindrome or not.

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

bool isPalindrome(char str[]) {
int left = 0;
int right = strlen(str) - 1;
while (left < right) {
if (str[left] != str[right]) {
return false;
}
left++;
right--;
}
return true;
```

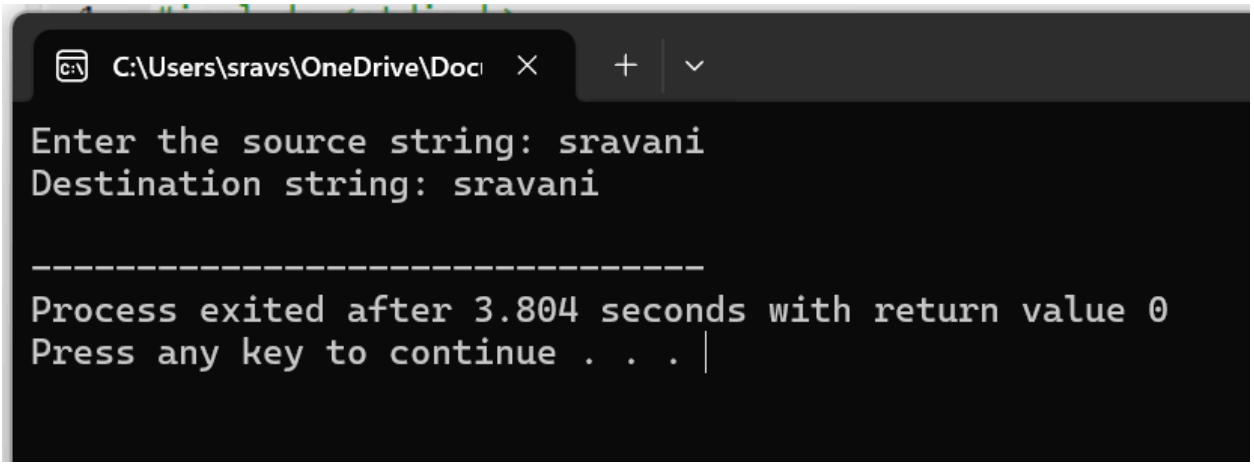
```
}
int main() {
char str[100];
printf("Enter a string: ");
scanf("%s", str);
if (isPalindrome(str)) {
printf("\"%s\" is a palindrome.\n", str);
} else {
printf("\"%s\" is not a palindrome.\n", str);
}
return 0;
}
```



11. Copy one string to another.

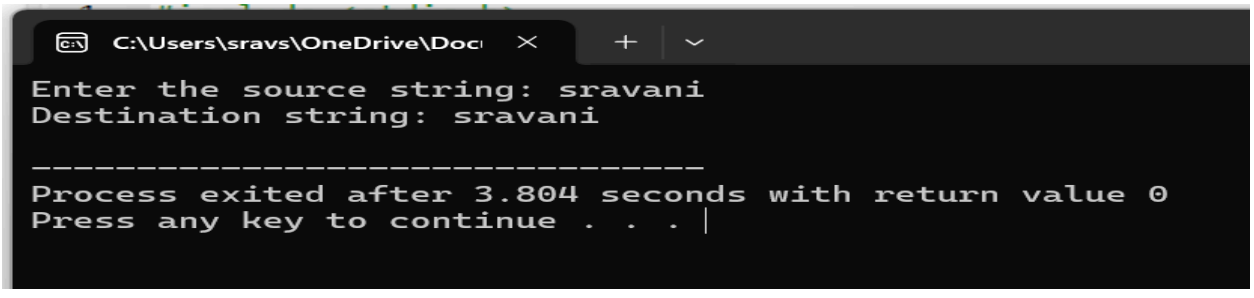
Using strcspn:

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

printf("Enter the source string: ");
fgets(source, sizeof(source), stdin);
source[strcspn(source, "\n")] = '\0';
strcpy(destination, source);
printf("Destination string: %s\n", destination);
return 0;
}
```



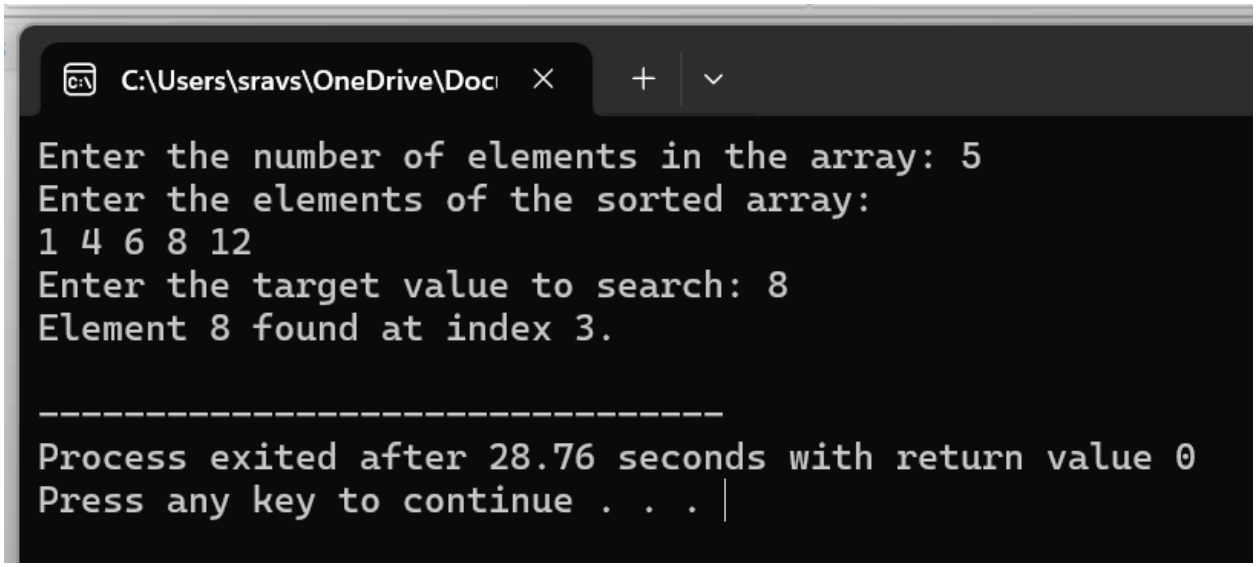
Manual copying:

```
#include <stdio.h>
void copyString(char *source, char *destination) {
while (*source != '\0') {
*destination = *source;
source++;
destination++;
}
*destination = '\0';
}
int main() {
char source[100], destination[100];
printf("Enter the source string: ");
fgets(source, sizeof(source), stdin);
source[strcspn(source, "\n")] = '\0';
copyString(source, destination);
printf("Destination string: %s\n", destination);
return 0;
}
```



12. Binary search

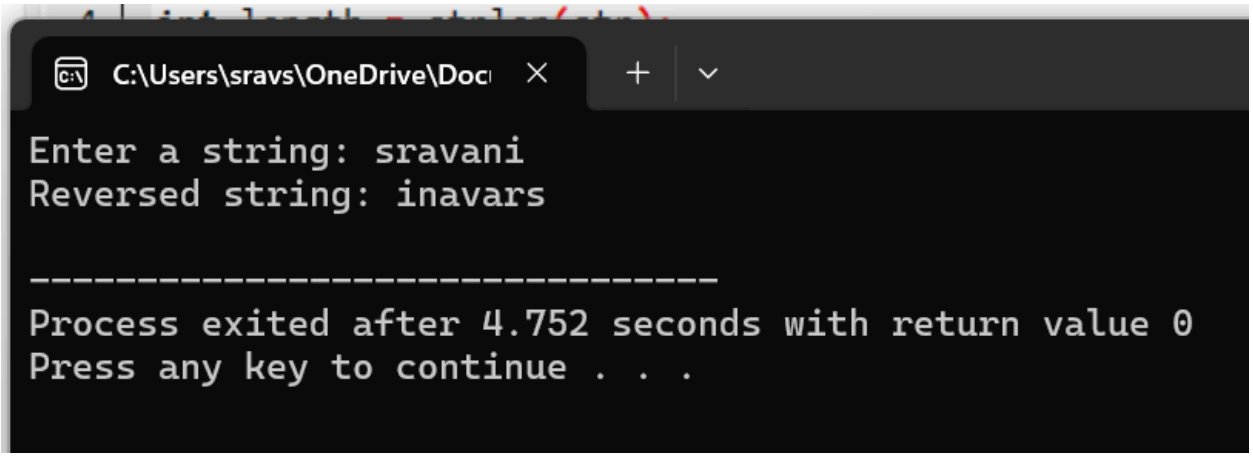
```
#include <stdio.h>
int binarySearch(int arr[], int size, int target) {
int low = 0;
int high = size - 1;
int mid;
while (low <= high) {
mid = low + (high - low) / 2;
if (arr[mid] == target) {
return mid;
}
if (arr[mid] > target) {
high = mid - 1;
}
else {
low = mid + 1;
}
}
return -1;
}
int main() {
int n, target, result;
printf("Enter the number of elements in the array: ");
scanf("%d", &n);
int arr[n];
printf("Enter the elements of the sorted array:\n");
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
printf("Enter the target value to search: ");
scanf("%d", &target);
result = binarySearch(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;
}
```



13. Reverse a string

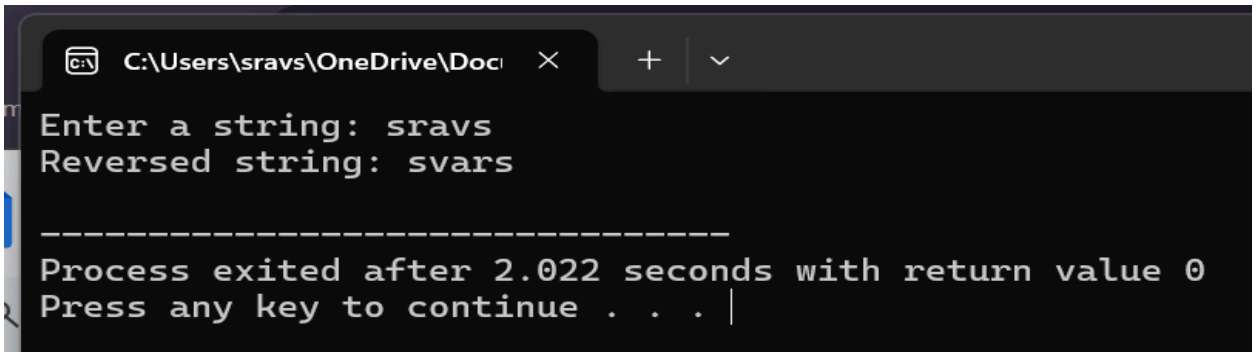
Using Temporary Array:

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

reverseString(str, reversed);
printf("Reversed string: %s\n", reversed);
return 0;
}
```



Using Loop:

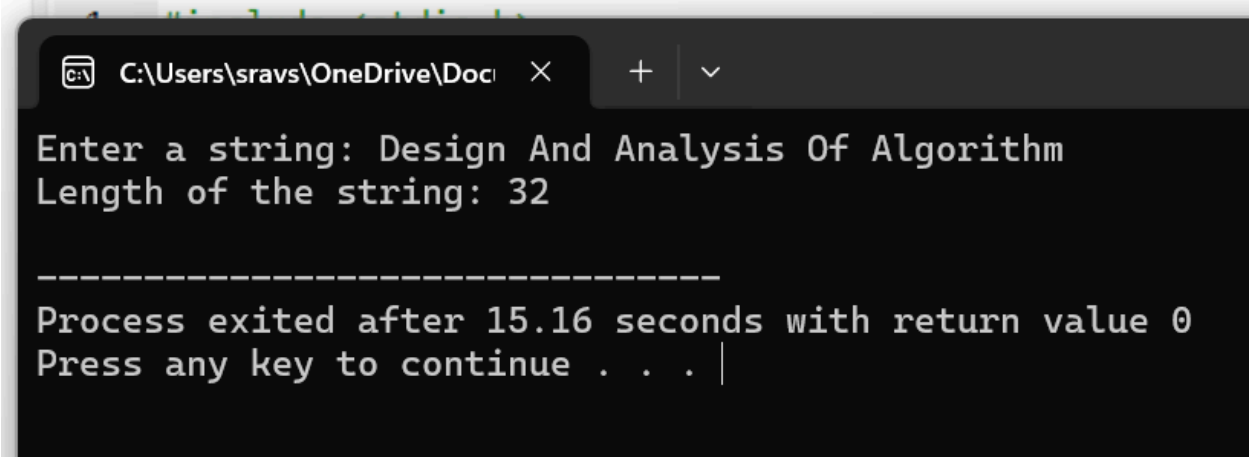
```
#include <stdio.h>
#include <string.h>
void printReverse(char str[]) {
int length = strlen(str);
for (int i = length - 1; i >= 0; i--) {
printf("%c", str[i]);
}
printf("\n");
}
int main() {
char str[100];
printf("Enter a string: ");
fgets(str, sizeof(str), stdin);
str[strcspn(str, "\n")] = '\0';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



14. Length of string

Using strlen Function:

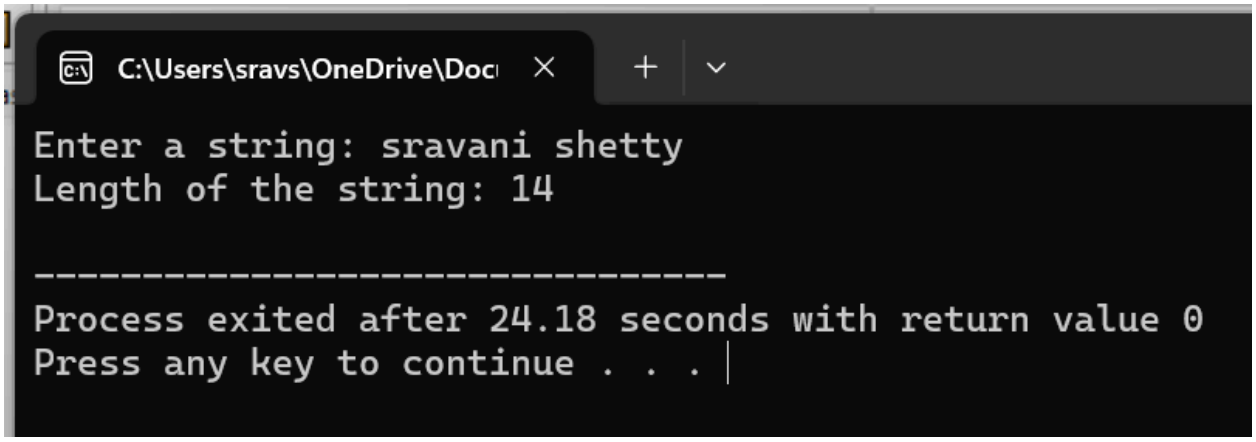
```
#include <stdio.h>
#include <string.h>
int main() {
char str[100];
printf("Enter a string: ");
fgets(str, sizeof(str), stdin);
str[strcspn(str, "\n")] = '\0';
int length = strlen(str);
printf("Length of the string: %d\n", length);
return 0;
}
```



Using a Custom Function:

```
#include <stdio.h>

int stringLength(char str[]) {
int length = 0;
while (str[length] != '\0') {
length++;
}
return length;
}
int main() {
char str[100];
printf("Enter a string: ");
fgets(str, sizeof(str), stdin);
str[strcspn(str, "\n")] = '\0';
```

```
int length = stringLength(str);
printf("Length of the string: %d\n", length);
return 0;
}
```



15. Strassen's Multiplication

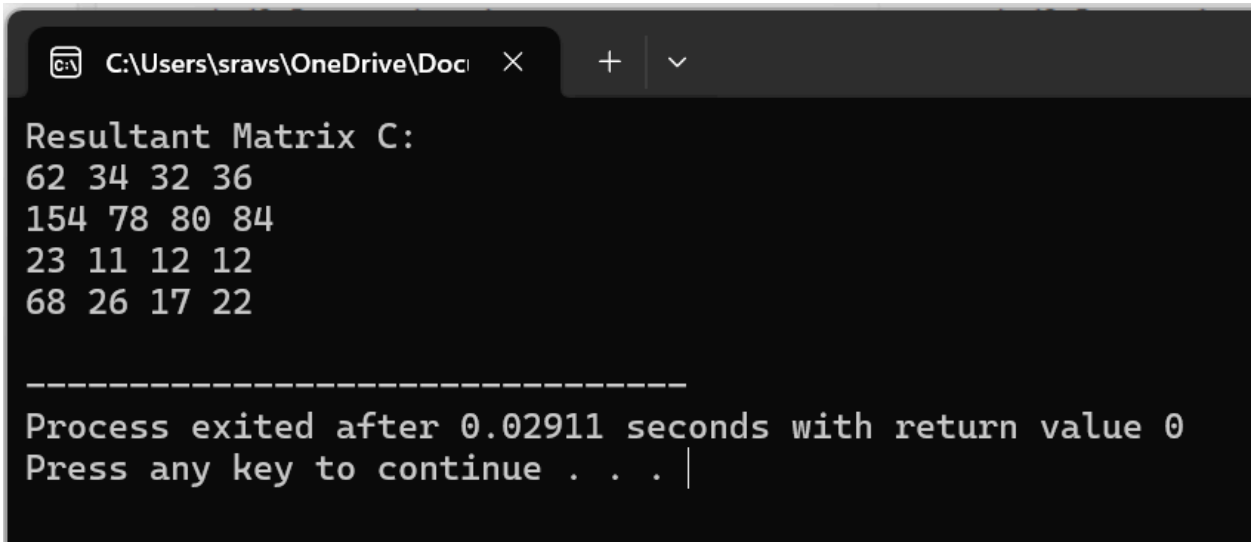
```
#include <stdio.h>
#include <stdlib.h>

int **allocateMatrix(int n) {
    int **matrix = (int **)malloc(n * sizeof(int *));
    for (int i = 0; i < n; i++) {
        matrix[i] = (int *)malloc(n * sizeof(int));
    }
    return matrix;
}

void freeMatrix(int **matrix, int n) {
    for (int i = 0; i < n; i++) {
        free(matrix[i]);
    }
    free(matrix);
}
void matrixAdd(int **A, int **B, int **C, 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 matrixSubtract(int **A, int **B, int **C, 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 strassenMultiply(int **A, int **B, int **C, int n) {
    if (n == 1) {
        C[0][0] = A[0][0] * B[0][0];
        return;
    }

    int newSize = n / 2;
    int **A11 = allocateMatrix(newSize);
    int **A12 = allocateMatrix(newSize);
    int **A21 = allocateMatrix(newSize);
    int **A22 = allocateMatrix(newSize);
    int **B11 = allocateMatrix(newSize);
    int **B12 = allocateMatrix(newSize);
    int **B21 = allocateMatrix(newSize);
    int **B22 = allocateMatrix(newSize);
    int **C11 = allocateMatrix(newSize);
    int **C12 = allocateMatrix(newSize);
    int **C21 = allocateMatrix(newSize);
    int **C22 = allocateMatrix(newSize);
    int **M1 = allocateMatrix(newSize);
    int **M2 = allocateMatrix(newSize);
    int **M3 = allocateMatrix(newSize);
    int **M4 = allocateMatrix(newSize);
    int **M5 = allocateMatrix(newSize);
    int **M6 = allocateMatrix(newSize);
    int **M7 = allocateMatrix(newSize);
    int **temp1 = allocateMatrix(newSize);
    int **temp2 = allocateMatrix(newSize);
    for (int i = 0; i < newSize; i++) {
        for (int j = 0; j < newSize; j++) {
```

```
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + newSize];
            A21[i][j] = A[i + newSize][j];
            A22[i][j] = A[i + newSize][j + newSize];

            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + newSize];
            B21[i][j] = B[i + newSize][j];
            B22[i][j] = B[i + newSize][j + newSize];
        }
    }
    matrixAdd(A11, A22, temp1, newSize);
    matrixAdd(B11, B22, temp2, newSize);
    strassenMultiply(temp1, temp2, M1, newSize);
    matrixAdd(A21, A22, temp1, newSize);
    strassenMultiply(temp1, B11, M2, newSize);
    matrixSubtract(B12, B22, temp1, newSize);
    strassenMultiply(A11, temp1, M3, newSize);
    matrixSubtract(B21, B11, temp1, newSize);
    strassenMultiply(A22, temp1, M4, newSize);
    matrixAdd(A11, A12, temp1, newSize);
    strassenMultiply(temp1, B22, M5, newSize);
    matrixSubtract(A21, A11, temp1, newSize);
    matrixAdd(B11, B12, temp2, newSize);
    strassenMultiply(temp1, temp2, M6, newSize);
    matrixSubtract(A12, A22, temp1, newSize);
    matrixAdd(B21, B22, temp2, newSize);
    strassenMultiply(temp1, temp2, M7, newSize);
    matrixAdd(M1, M4, temp1, newSize);
    matrixSubtract(temp1, M5, temp2, newSize);
    matrixAdd(temp2, M7, C11, newSize);
    matrixAdd(M3, M5, C12, newSize);
    matrixAdd(M2, M4, C21, newSize);
    matrixSubtract(M1, M2, temp1, newSize);
    matrixAdd(temp1, M3, temp2, newSize);
    matrixAdd(temp2, M6, C22, newSize);
    for (int i = 0; i < newSize; i++) {
        for (int j = 0; j < newSize; j++) {
            C[i][j] = C11[i][j];
            C[i][j + newSize] = C12[i][j];
```

```
            C[i + newSize][j] = C21[i][j];
            C[i + newSize][j + newSize] = C22[i][j];
        }
    }

    freeMatrix(A11, newSize); freeMatrix(A12, newSize); freeMatrix(A21, newSize);
freeMatrix(A22, newSize);
    freeMatrix(B11, newSize); freeMatrix(B12, newSize); freeMatrix(B21, newSize);
freeMatrix(B22, newSize);
    freeMatrix(C11, newSize); freeMatrix(C12, newSize); freeMatrix(C21, newSize);
freeMatrix(C22, newSize);
    freeMatrix(M1, newSize); freeMatrix(M2, newSize); freeMatrix(M3, newSize);
freeMatrix(M4, newSize);
    freeMatrix(M5, newSize); freeMatrix(M6, newSize); freeMatrix(M7, newSize);
freeMatrix(temp1, newSize); freeMatrix(temp2, newSize);
}
void printMatrix(int **matrix, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}
int main() {
    int n = 4;
    int **A = allocateMatrix(n);
    int **B = allocateMatrix(n);
    int **C = allocateMatrix(n);
    int sampleA[4][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {1, 1, 1, 1},
        {1, 1, 1, 6}
    };

    int sampleB[4][4] = {
        {7, 0, 1, 0},
        {2, 2, 3, 2},
        {5, 6, 7, 8},
```

```
        {9, 3, 1, 2}
    };
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            A[i][j] = sampleA[i][j];
            B[i][j] = sampleB[i][j];
        }
    }
    strassenMultiply(A, B, C, n);
    printf("Resultant Matrix C:\n");
    printMatrix(C, n);
    freeMatrix(A, n);
    freeMatrix(B, n);
    freeMatrix(C, n);

    return 0;
}
```

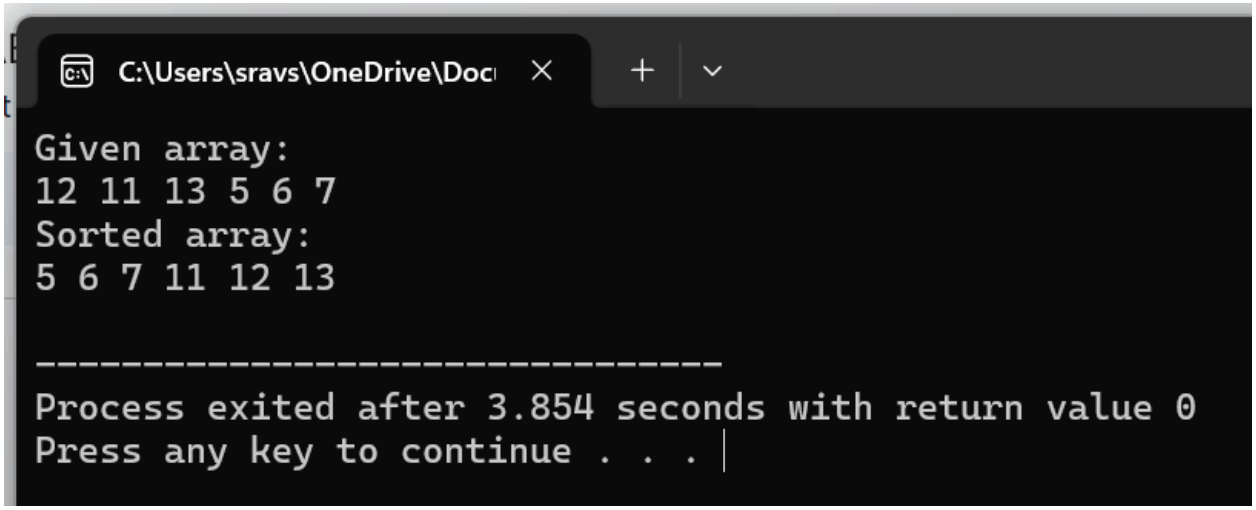


16. Merge sort

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int left, int mid, int right) {
int n1 = mid - left + 1;
int n2 = right - mid;
int* L = (int*)malloc(n1 * sizeof(int));
```

```
int* R = (int*)malloc(n2 * sizeof(int));
    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;
    int j = 0;
    int k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }
    while (i < n1) {
        arr[k++] = L[i++];
    }
    while (j < n2) {
        arr[k++] = R[j++];
    }
    free(L);
    free(R);
}
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 size = sizeof(arr) / sizeof(arr[0]);
printf("Given array:\n");
printArray(arr, size);
mergeSort(arr, 0, size - 1);
printf("Sorted array:\n");
printArray(arr, size);
return 0;
}
```

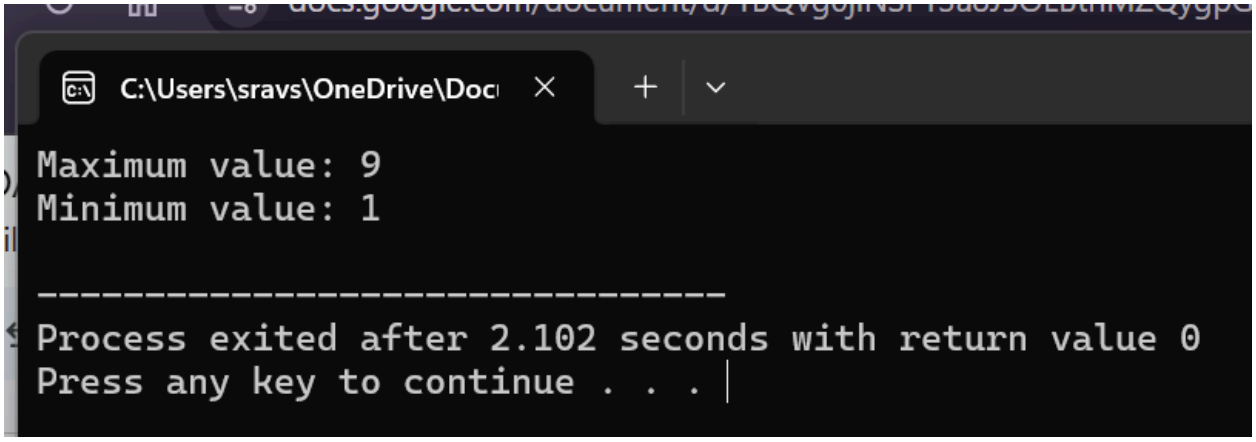


DAY- 03

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

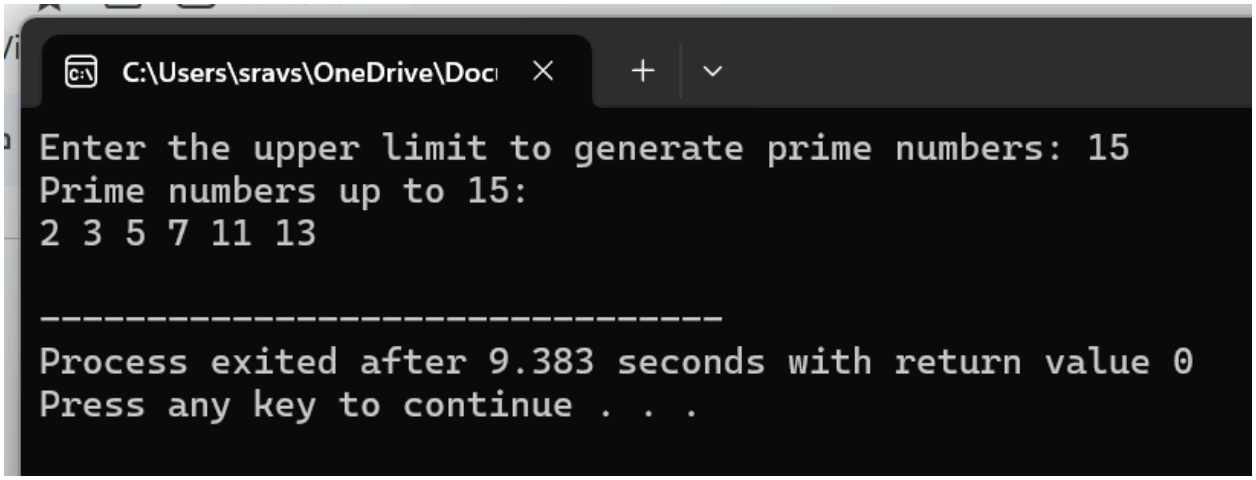
```
#include <stdio.h>
#include <limits.h>

void findMaxMin(int arr[], int left, int right, int *max, int *min) {
    if (left == right) {
        *max = arr[left];
        *min = arr[left];
```

```
    } else if (right == left + 1) {
        if (arr[left] > arr[right]) {
            *max = arr[left];
            *min = arr[right];
        } else {
            *max = arr[right];
            *min = arr[left];
        }
    } else {
        int mid = left + (right - left) / 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[] = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
    int size = sizeof(arr) / sizeof(arr[0]);
    int max, min;
    findMaxMin(arr, 0, size - 1, &max, &min);
    printf("Maximum value: %d\n", max);
    printf("Minimum value: %d\n", min);

    return 0;
}
```



18. Generate all the prime numbers.

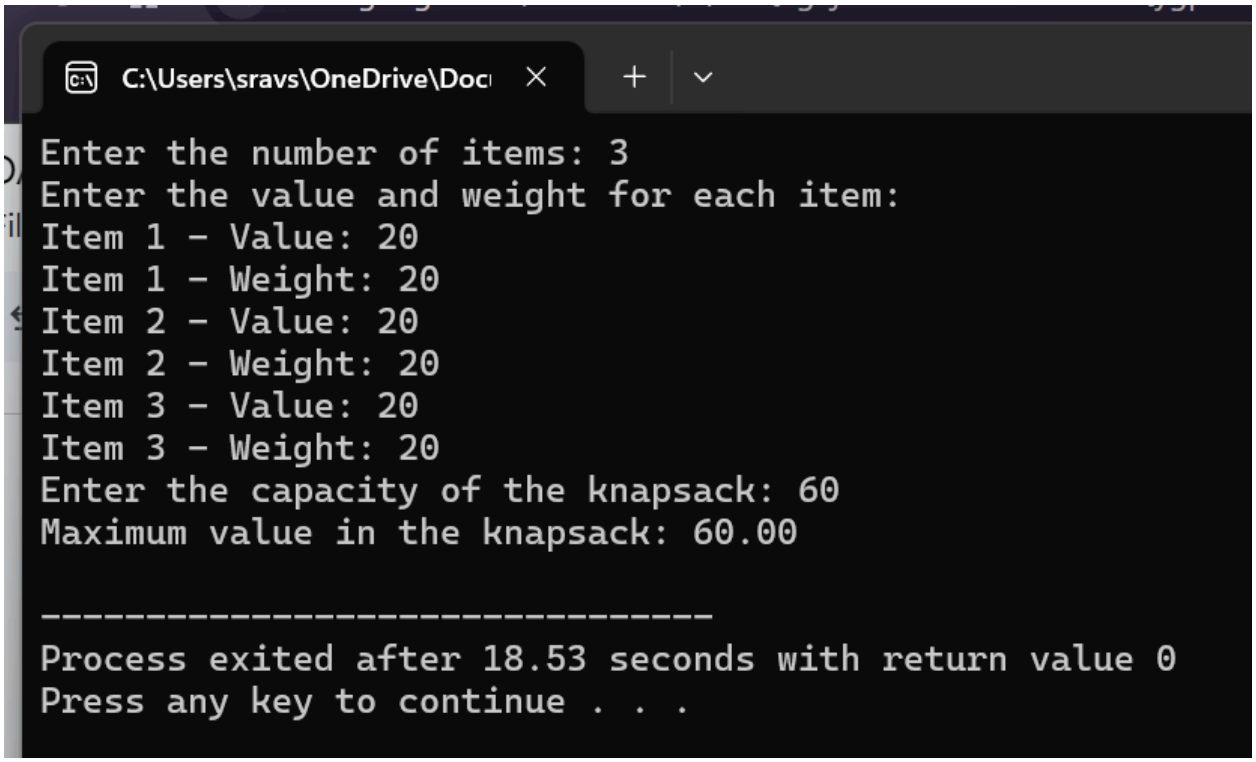
```
#include <stdio.h>
#include <stdbool.h>
void sieveOfEratosthenes(int n) {
bool prime[n + 1];
for (int i = 0; i <= n; i++) {
prime[i] = true;
}
prime[0] = prime[1] = false;
for (int p = 2; p * p <= n; p++) {

if (prime[p] == true) {

for (int i = p * p; i <= n; i += p) {
prime[i] = false;
}
}
}
printf("Prime numbers up to %d:\n", n);
for (int p = 2; p <= n; p++) {
if (prime[p]) {
printf("%d ", p);
}
}
printf("\n");
}
int main() {
int n;
printf("Enter the upper limit to generate prime numbers: ");
scanf("%d", &n);
sieveOfEratosthenes(n);
return 0;
}
```



19. Knapsack problem using greedy techniques.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
    int value;
    int weight;
    float ratio;
} Item;
int compare(const void *a, const void *b) {
    Item *item1 = (Item *)a;
    Item *item2 = (Item *)b;
    return (item2->ratio > item1->ratio) - (item2->ratio < item1->ratio);
}
float knapsack(Item items[], int n, int capacity) {
    qsort(items, n, sizeof(Item), compare);
    int currentWeight = 0;
    float totalValue = 0.0;
    for (int i = 0; i < n; i++) {
        if (currentWeight + items[i].weight <= capacity) {
            currentWeight += items[i].weight;
            totalValue += items[i].value;
        } else {
            int remaining = capacity - currentWeight;
            totalValue += items[i].value * ((float)remaining / items[i].weight);
            break;
        }
    }
```

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

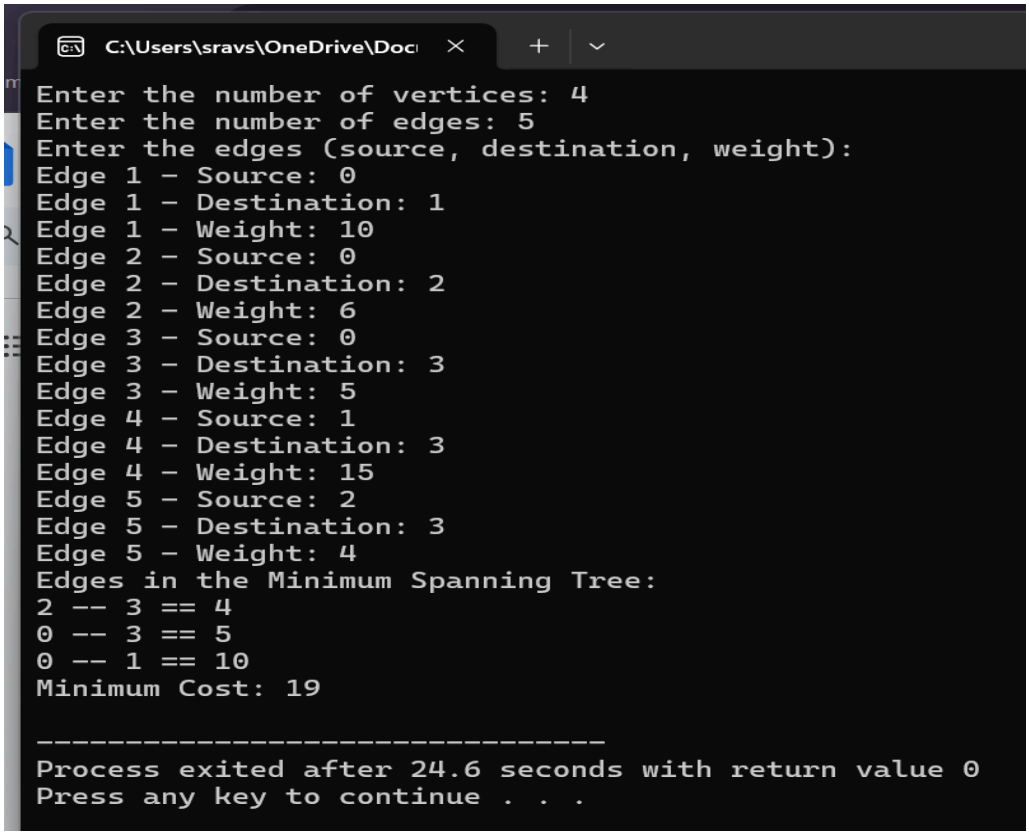


20. MST using greedy techniques.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct {
int src, dest, weight;
} Edge;
typedef struct {
int parent, rank;
} Subset;
int compareEdges(const void *a, const void *b) {
Edge *edge1 = (Edge *)a;
Edge *edge2 = (Edge *)b;
return edge1->weight - edge2->weight;
}
int find(Subset subsets[], int i) {
    if (subsets[i].parent != i) {
        subsets[i].parent = find(subsets, subsets[i].parent);
    }
    return subsets[i].parent;
}
void unionSubsets(Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank) {
        subsets[xroot].parent = yroot;
    } else if (subsets[xroot].rank > subsets[yroot].rank) {
        subsets[yroot].parent = xroot;
    } else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}
void kruskal(int vertices, Edge edges[], int e) {
    Edge result[MAX];
    Subset subsets[vertices];
    qsort(edges, e, sizeof(Edge), compareEdges);
    for (int i = 0; i < vertices; i++) {
```

```
        subsets[i].parent = i;
        subsets[i].rank = 0;
    }

    int edgeIndex = 0;
    int i = 0;

    while (edgeIndex < vertices - 1 && i < e) {
        Edge nextEdge = edges[i++];

        int x = find(subsets, nextEdge.src);
        int y = find(subsets, nextEdge.dest);
        if (x != y) {
            result[edgeIndex++] = nextEdge;
            unionSubsets(subsets, x, y);
        }
    }
    printf("Edges in the Minimum Spanning Tree:\n");
    int minimumCost = 0;
    for (int i = 0; i < edgeIndex; i++) {
printf("%d -- %d == %d\n", result[i].src, result[i].dest,
result[i].weight);
minimumCost += result[i].weight;
}
printf("Minimum Cost: %d\n", minimumCost);
}
int main() {
int vertices, edgesCount;
printf("Enter the number of vertices: ");
scanf("%d", &vertices);
printf("Enter the number of edges: ");
scanf("%d", &edgesCount);
Edge edges[edgesCount];
printf("Enter the edges (source, destination, weight):\n");
for (int i = 0; i < edgesCount; i++) {
printf("Edge %d - Source: ", i + 1);
scanf("%d", &edges[i].src);
printf("Edge %d - Destination: ", i + 1);
scanf("%d", &edges[i].dest);
printf("Edge %d - Weight: ", i + 1);
```

```
scanf("%d", &edges[i].weight);
}
kruskal(vertices, edges, edgesCount);
return 0;
}
```

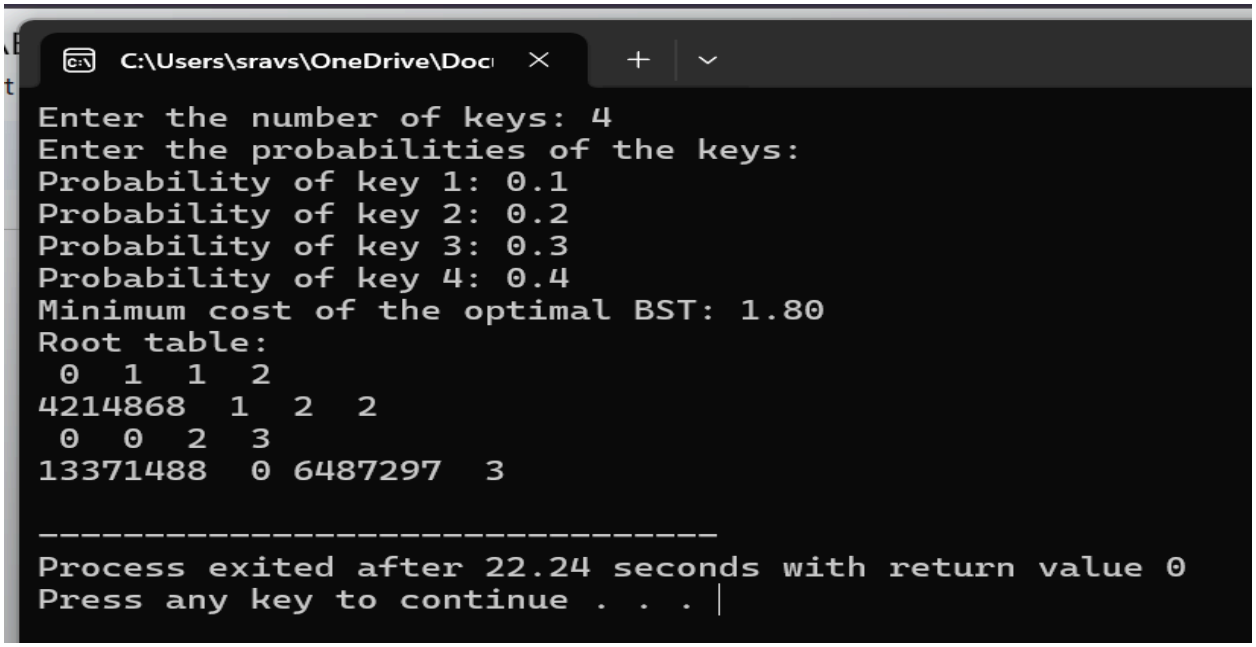


21. .Using Dynamic programming concept to find out Optimal binary search tree.

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

void optimalBST(float p[], int n) {
    float e[n][n], w[n][n];
    int root[n][n];

    for (int i = 0; i < n; i++) {
        e[i][i] = p[i];
        w[i][i] = p[i];
        root[i][i] = i;
```

```
    }

    for (int len = 2; len <= n; len++) {
        for (int i = 0; i <= n - len; i++) {
            int j = i + len - 1;
            e[i][j] = INT_MAX;
            w[i][j] = w[i][j - 1] + p[j];

            for (int k = i; k <= j; k++) {
                float t = (k > i ? e[i][k - 1] : 0) + (k < j ? e[k + 1][j]
: 0) + w[i][j];
                if (t < e[i][j]) {
                    e[i][j] = t;
                    root[i][j] = k;
                }
            }
        }
    }

    printf("Minimum cost of the optimal BST: %.2f\n", e[0][n - 1]);

    printf("Root table:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%2d ", root[i][j]);
        }
        printf("\n");
    }
}

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

    float p[n];
    printf("Enter the probabilities of the keys:\n");
    for (int i = 0; i < n; i++) {
        printf("Probability of key %d: ", i + 1);
        scanf("%f", &p[i]);
```

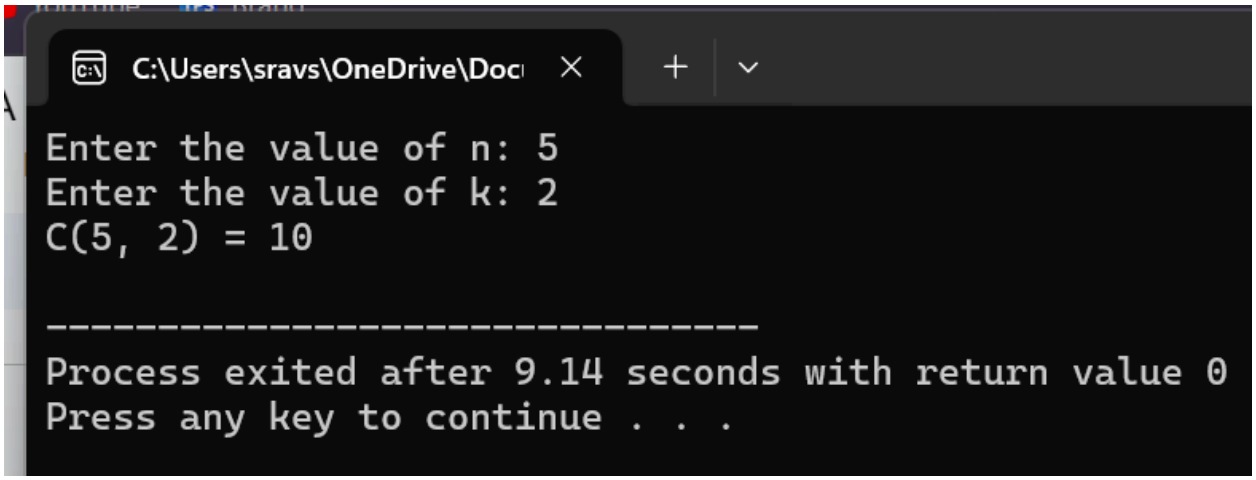
```
    }
    optimalBST(p, n);

    return 0;
}
```



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

```
#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;
} else {
C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
}
}
}
return C[n][k];
}
int main() {
int n, k;
```

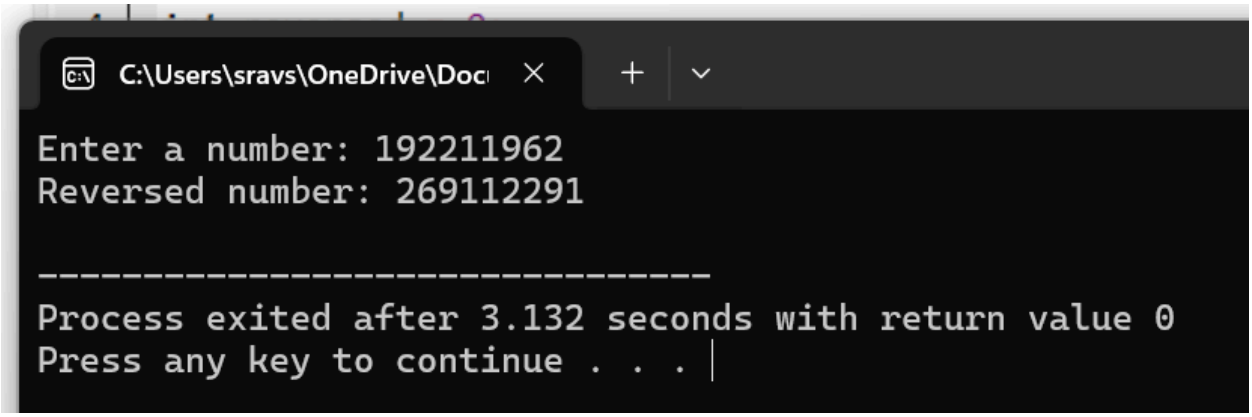
```
printf("Enter the value of n: ");
scanf("%d", &n);
printf("Enter the value of k: ");
scanf("%d", &k);
if (k > n || k < 0) {
printf("Invalid values for n and k.\n");
} else {
printf("C(%d, %d) = %d\n", n, k, binomialCoefficient(n, k));
}
return 0;
}
```



23. Reverse of a given number.

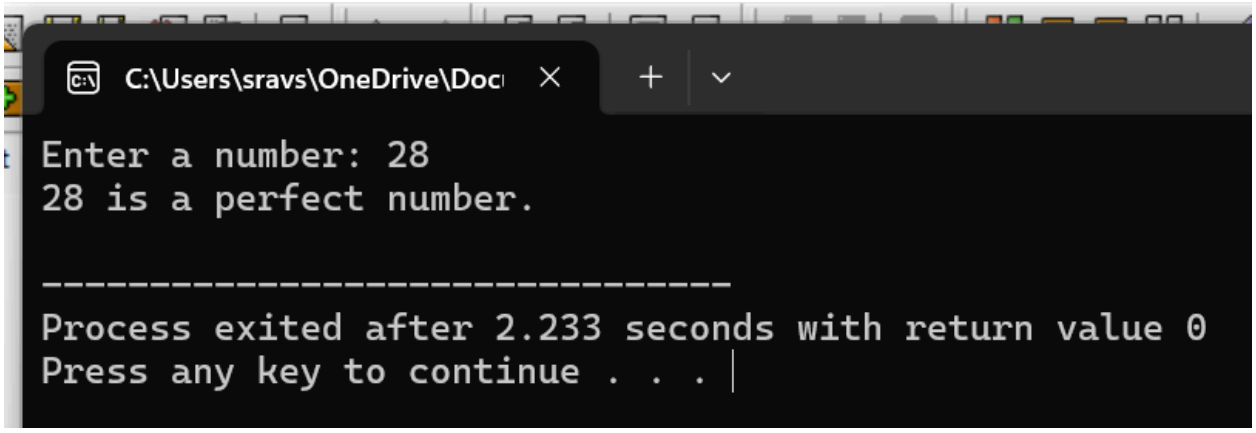
```
#include <stdio.h>
// Function to reverse the digits of a number
int reverseNumber(int num) {
int reversed = 0;
while (num != 0) {
int digit = num % 10;
reversed = reversed * 10 + digit;
num /= 10;
}
return reversed;
}
int main() {
int number;
printf("Enter a number: ");
scanf("%d", &number);
```

```
int reversedNumber = reverseNumber(number);
printf("Reversed number: %d\n", reversedNumber);
return 0;
}
```



24. Perfect number.

```
#include <stdio.h>
int isPerfectNumber(int num) {
if (num <= 1) return 0;
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;
}
```

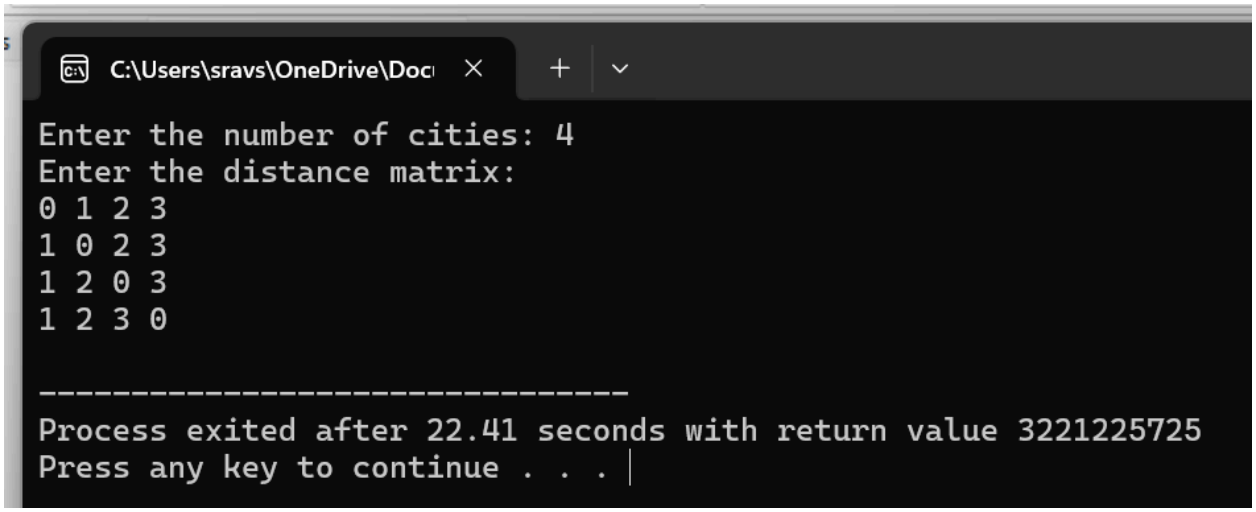


DAY- 04

25. Traveling salesman problem using dynamic programming.

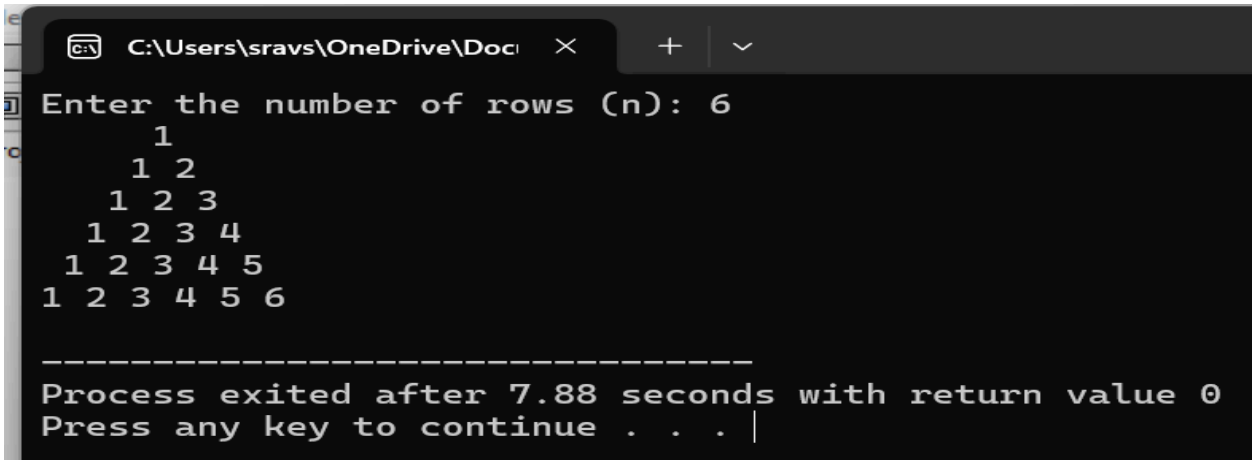
```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>
#define MAX 16
#define INF INT_MAX

int tsp(int n, int dist[MAX][MAX]) {
int dp[1 << MAX][MAX];
   for (int mask = 0; mask < (1 << n); mask++) {
      for (int i = 0; i < n; i++) {
         dp[mask][i] = INF;
      }
   }
   dp[1][0] = 0;
   for (int mask = 1; mask < (1 << n); mask += 2) {
      for (int u = 0; u < n; u++) {
         if (!(mask & (1 << u))) continue;
         for (int v = 0; v < n; v++) {
            if (mask & (1 << v)) continue;
            int newMask = mask | (1 << v);
```

```
            dp[newMask][v] = (dp[newMask][v] < dp[mask][u] +
dist[u][v]) ? dp[newMask][v] : dp[mask][u] + dist[u][v];
          }
       }
   }
   int answer = INF;
   for (int i = 1; i < n; i++) {
      answer = (answer < dp[(1 << n) - 1][i] + dist[i][0]) ? answer :
dp[(1 << n) - 1][i] + dist[i][0];
   }

   return answer;
}

int main() {
   int n;
   printf("Enter the number of cities: ");
   scanf("%d", &n);
   int dist[MAX][MAX];
   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]);
      }
   }
   int result = tsp(n, dist);
   printf("The minimum cost of the TSP is: %d\n", result);

   return 0;
}
```



26. Right angled triangle Format.

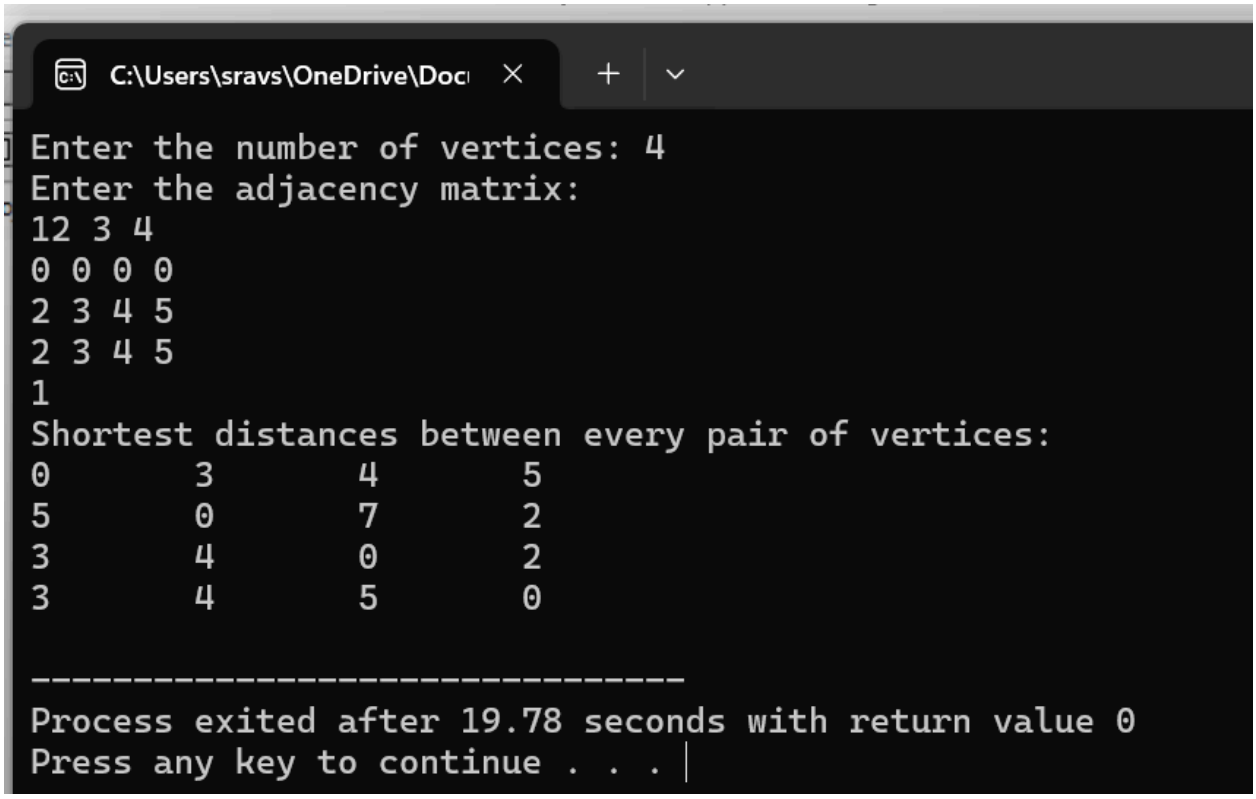
```
#include <stdio.h>
void printPattern(int n) {
    for (int i = 1; i <= n; i++) {
        for (int j = 0; j < n - i; j++) {
            printf(" ");
        }
        for (int k = 1; k <= i; k++) {
            printf("%d ", k);
        }
        printf("\n");
    }
}
int main() {
    int n;
    printf("Enter the number of rows (n): ");
    scanf("%d", &n);
    printPattern(n);
    return 0;
}
```



27. Floyd's Warshall 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];
            }
        }
    }
```

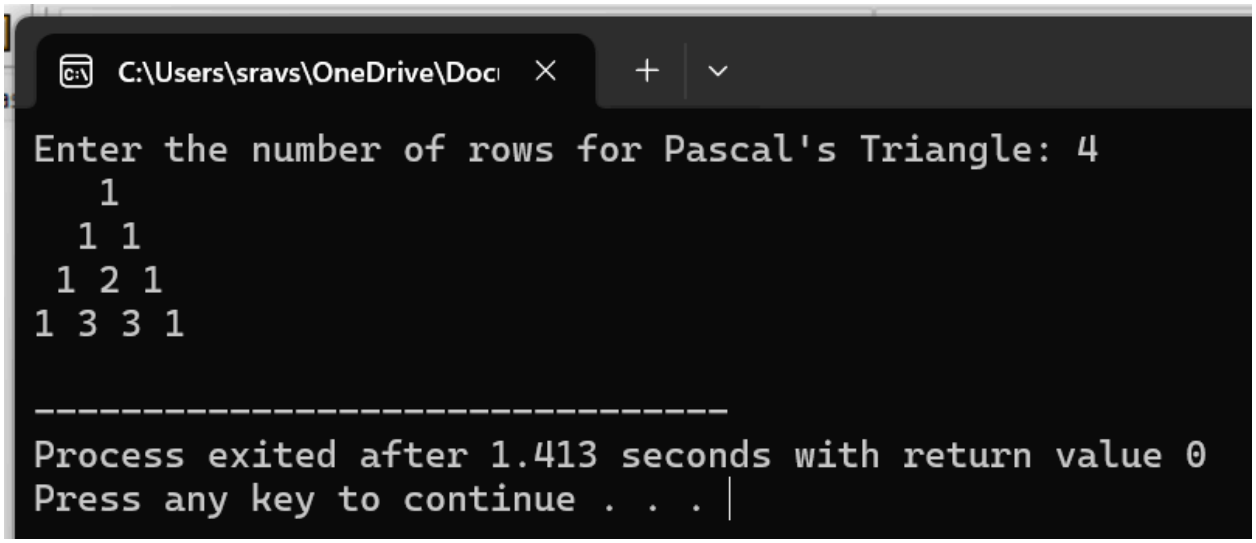
```
    }
    printf("Shortest distances between every pair of vertices:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dist[i][j] == INF) {
                printf("INF\t");
            } else {
                printf("%d\t", dist[i][j]);
            }
 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



28. Pascal triangle.

```
#include <stdio.h>
void printPascalsTriangle(int n) {
int triangle[n][n];
for (int i = 0; i < n; i++) {
for (int j = 0; j <= i; j++) {
if (j == 0 || j == i) {
triangle[i][j] = 1;
} else {
triangle[i][j] = triangle[i - 1][j - 1] + triangle[i -
1][j];
}
}
}
for (int i = 0; i < n; i++) {
for (int j = 0; j < n - i - 1; j++) {
printf(" ");
}
for (int j = 0; j <= i; j++) {
```

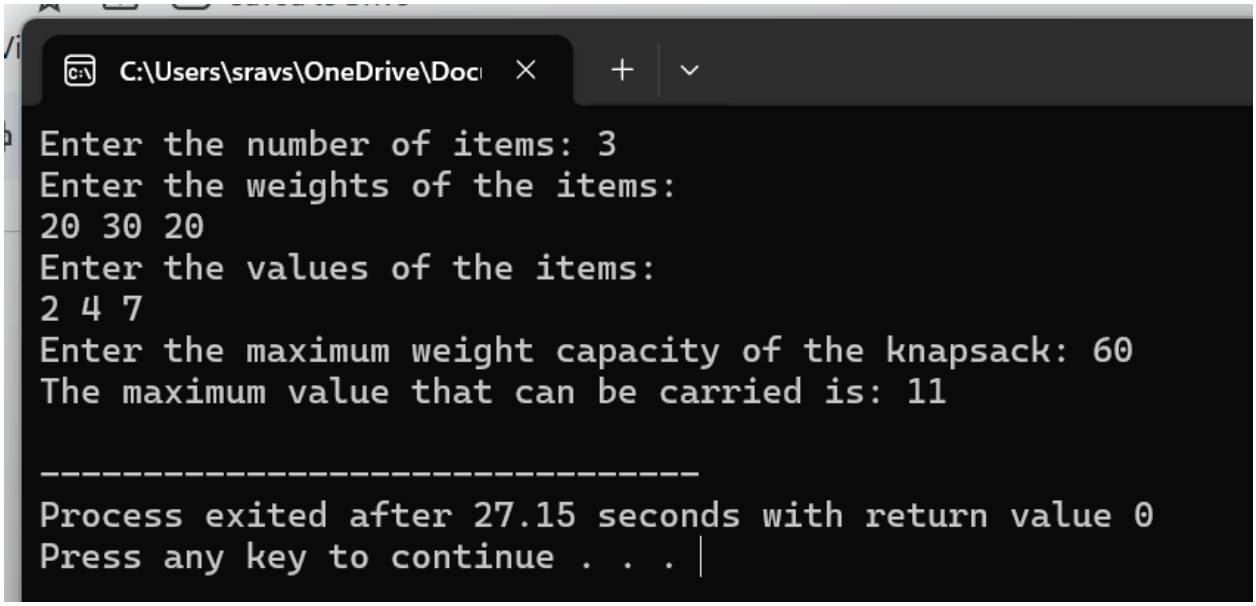
```
printf("%d ", triangle[i][j]);
}
printf("\n");
}
}
int main() {
int n;
printf("Enter the number of rows for Pascal's Triangle: ");
scanf("%d", &n);
printPascalsTriangle(n);
return 0;
}
```



29. Find the optimal cost by using the appropriate algorithm.

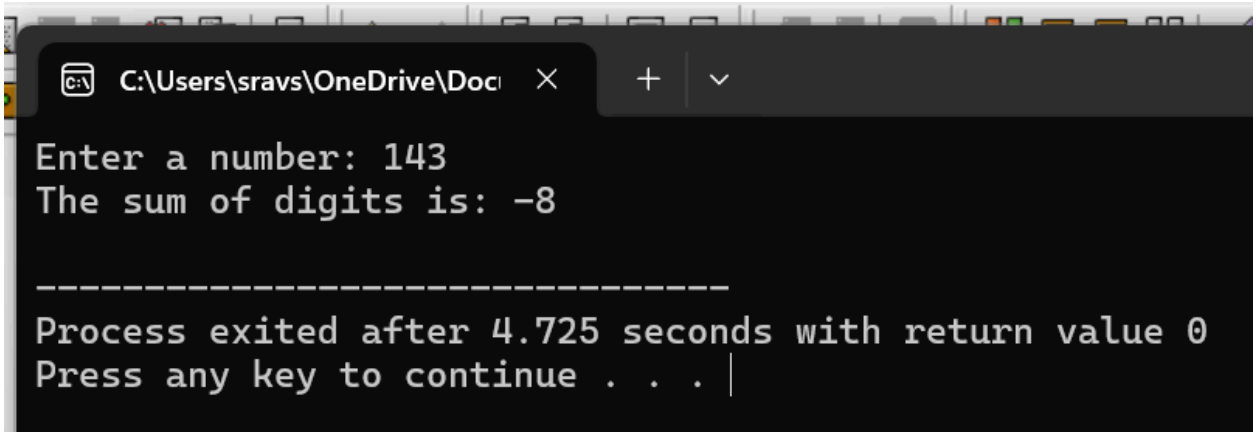
```
#include <stdio.h>
int knapsack(int W, int weights[], int values[], int n) {
   int dp[n + 1][W + 1];
   for (int i = 0; i <= n; i++) {
      for (int w = 0; w <= W; w++) {
         if (i == 0 || w == 0) {
            dp[i][w] = 0;
         } else if (weights[i - 1] <= w) {
            dp[i][w] = (values[i - 1] + dp[i - 1][w - weights[i - 1]] >
dp[i - 1][w]) ?
                     (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][W];
}

int main() {
    int n, W;
    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 maximum weight capacity of the knapsack: ");
    scanf("%d", &W);
    int result = knapsack(W, weights, values, n);
    printf("The maximum value that can be carried is: %d\n", result);

    return 0;
}
```



30. Sum of digits.

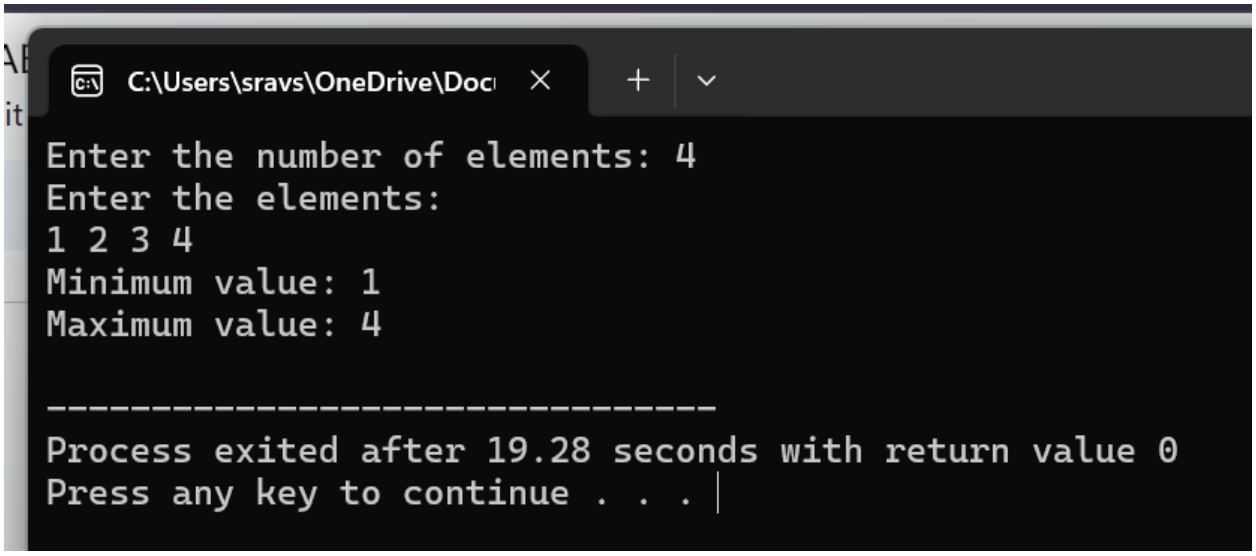
```
#include <stdio.h>
int sumOfDigits(int num) {
int sum = 0;
while (num != 0) {
sum += num % 10;
num /= 10;
}
return sum;
}
int main() {
int number;
printf("Enter a number: ");
scanf("%d", &number);
if (number < 0) {
}
number = -number;
int result = sumOfDigits(number);
printf("The sum of digits is: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



31. Print a minimum and maximum value sequence for all the numbers in a list.

```
#include <stdio.h>
void findMinMax(int arr[], int size, int *min, int *max) {
    *min = arr[0];
    *max = arr[0];

    for (int i = 1; i < size; i++) {
        if (arr[i] < *min) {
            *min = arr[i];
        }
        if (arr[i] > *max) {
            *max = arr[i];
        }
    }
}

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

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

```
    int min, max;
    findMinMax(arr, n, &min, &max);
    printf("Minimum value: %d\n", min);
    printf("Maximum value: %d\n", max);

    return 0;
}
```

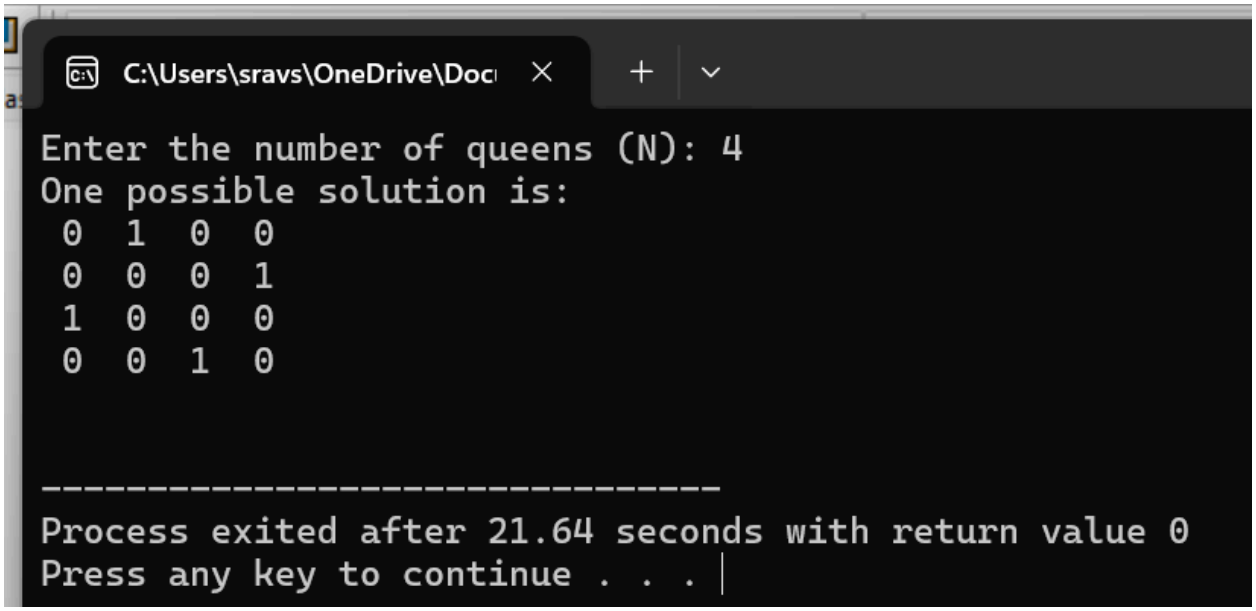


32. N- Queen problem using Backtracking.

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

#define MAX 20
void printSolution(int board[MAX][MAX], int N) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf(" %d ", board[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
bool isSafe(int board[MAX][MAX], int row, int col, int N) {
    for (int i = 0; i < row; i++) {
        if (board[i][col]) {
```

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

    return true;
}
bool solveNQueens(int board[MAX][MAX], int row, int N) {
    if (row >= N) {
        return true;
    }

    for (int col = 0; col < N; col++) {
        if (isSafe(board, row, col, N)) {
            board[row][col] = 1;
            if (solveNQueens(board, row + 1, N)) {
                return true;
            }
            board[row][col] = 0;
        }
    }

    return false;
}

int main() {
    int N;
    int board[MAX][MAX] = {0};
    printf("Enter the number of queens (N): ");
    scanf("%d", &N);
    if (solveNQueens(board, 0, N)) {
```

```
        printf("One possible solution is:\n");
        printSolution(board, N);
    } else {
        printf("No solution exists for N = %d\n", N);
    }

    return 0;
}
```

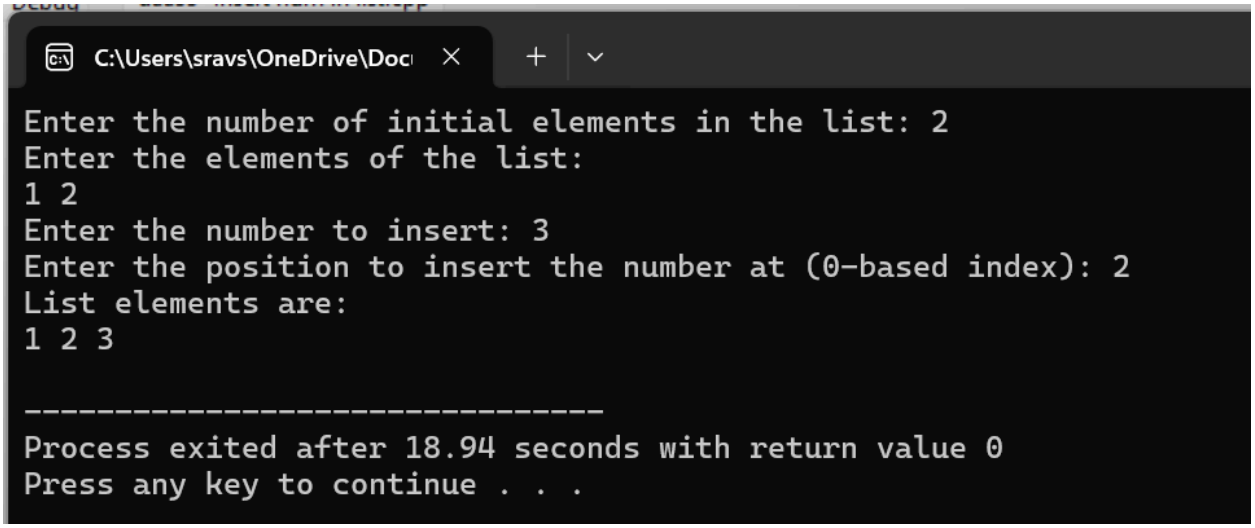


DAY- 05

33. Insert a number in a list.

```
#include <stdio.h>
#define MAX 100
void insertNumber(int list[], int *size, int number, int position) {
if (position < 0 || position > *size) {
printf("Invalid position!\n");
return;
}
if (*size >= MAX) {
printf("List is full!\n");
return;
}
```

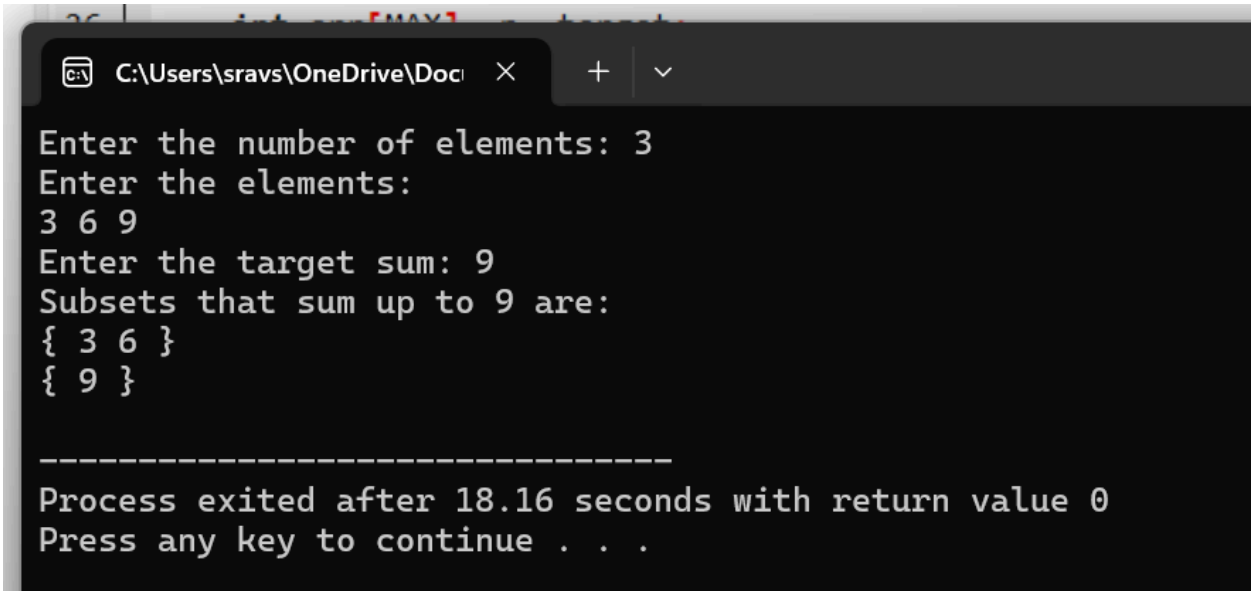
```
for (int i = *size; i > position; i--) {
list[i] = list[i - 1];
}
list[position] = number;
(*size)++;
}
void printList(int list[], int size) {
printf("List elements are:\n");
for (int i = 0; i < size; i++) {
printf("%d ", list[i]);
}
printf("\n");
}
int main() {
int list[MAX];
int size = 0;
int number, position;
printf("Enter the number of initial elements in the list: ");
scanf("%d", &size);
printf("Enter the elements of the list:\n");
for (int i = 0; i < size; i++) {
scanf("%d", &list[i]);
}
printf("Enter the number to insert: ");
scanf("%d", &number);
printf("Enter the position to insert the number at (0-based index): ");
scanf("%d", &position);
insertNumber(list, &size, number, position);
printList(list, size);
return 0;
}
```



34. Sum of subsets problem using backtracking.

```
#include <stdio.h>
#define MAX 20
void printSubset(int subset[], int size) {
    printf("{ ");
    for (int i = 0; i < size; i++) {
        printf("%d ", subset[i]);
    }
    printf("}\n");
}
void findSubsets(int arr[], int n, int index, int target, int currentSum,
int subset[], int subsetSize) {
    if (currentSum == target) {
        printSubset(subset, subsetSize);
        return;
    }
    if (index >= n || currentSum > target) {
        return;
    }
    subset[subsetSize] = arr[index];
    findSubsets(arr, n, index + 1, target, currentSum + arr[index], subset,
subsetSize + 1);
    findSubsets(arr, n, index + 1, target, currentSum, subset, subsetSize);
}
```

```
int main() {
    int arr[MAX], n, target;
    int subset[MAX];
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter the elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the target sum: ");
    scanf("%d", &target);
    printf("Subsets that sum up to %d are:\n", target);
    findSubsets(arr, n, 0, target, 0, subset, 0);

    return 0;
}
```

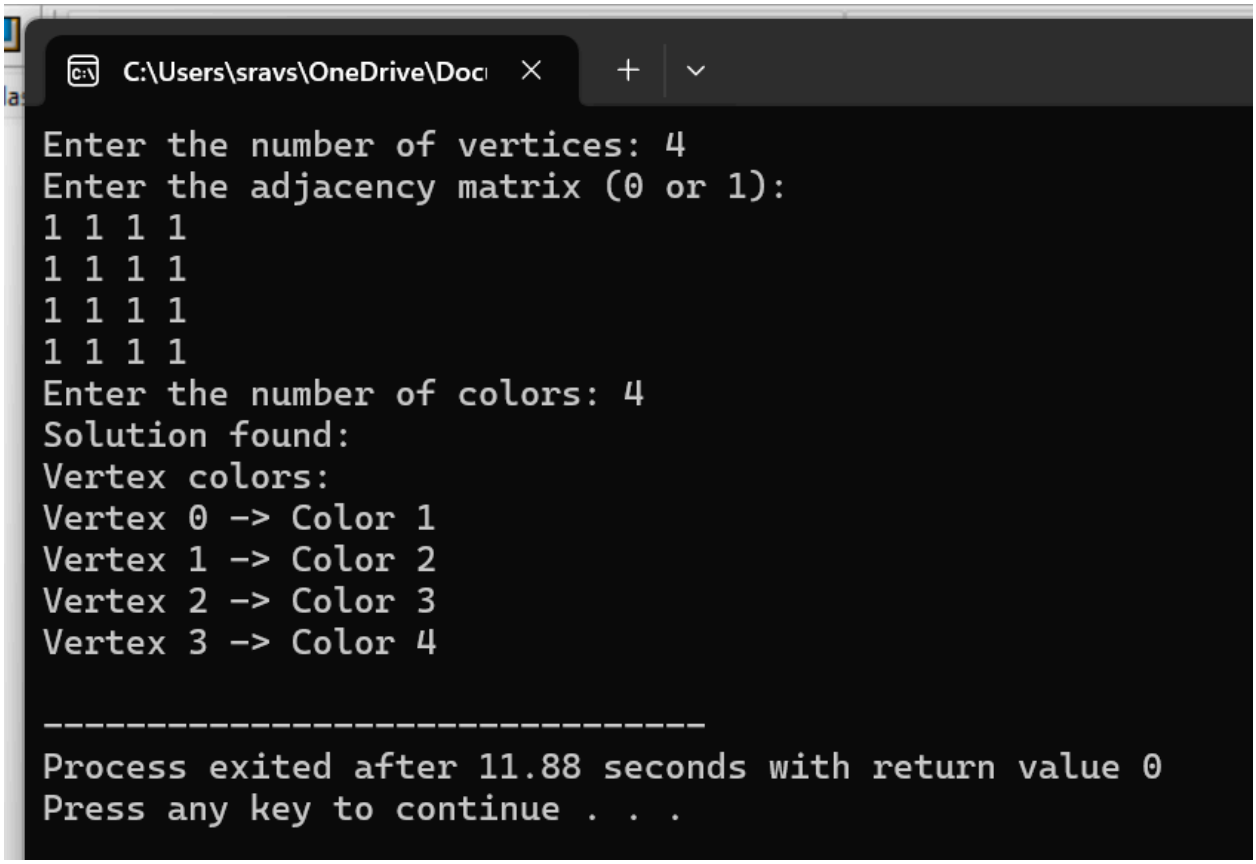


35. Graph coloring using Backtracking.

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

void printSolution(int color[], int V);

bool isSafe(int v, bool graph[][20], int color[], int c, int V)
{
```

```
    for (int i = 0; i < V; i++)
        if (graph[v][i] && c == color[i])
            return false;
    return true;
}

bool graphColoringUtil(bool graph[][20], int m, int color[], int v, int V)
{
    if (v == V)
        return true;

    for (int c = 1; c <= m; c++)
    {
        if (isSafe(v, graph, color, c, V))
        {
            color[v] = c;
            if (graphColoringUtil(graph, m, color, v + 1, V))
                return true;
            color[v] = 0;
        }
    }

    return false;
}

bool graphColoring(bool graph[][20], int m, int V)
{
    int color[20];
    for (int i = 0; i < V; i++)
        color[i] = 0;

    if (!graphColoringUtil(graph, m, color, 0, V))
    {
        printf("Solution does not exist\n");
        return false;
    }

    printf("Solution found:\n");
    printSolution(color, V);
    return true;
```

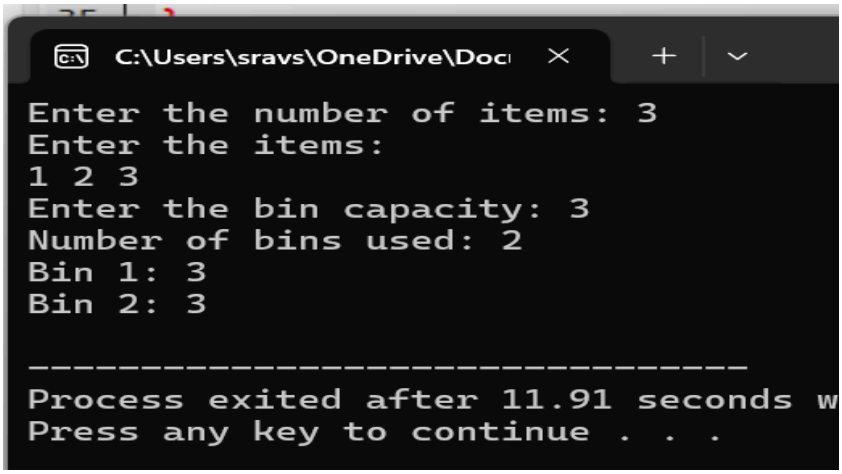
```
}

void printSolution(int color[], int V)
{
   printf("Vertex colors:\n");
   for (int i = 0; i < V; i++)
      printf("Vertex %d -> Color %d\n", i, color[i]);
}

int main()
{
   int V, m;
   bool graph[20][20];
   printf("Enter the number of vertices: ");
   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", (int *)&graph[i][j]);

   printf("Enter the number of colors: ");
   scanf("%d", &m);

   graphColoring(graph, m, V);

   return 0;
}
```



36. Container loader problem.

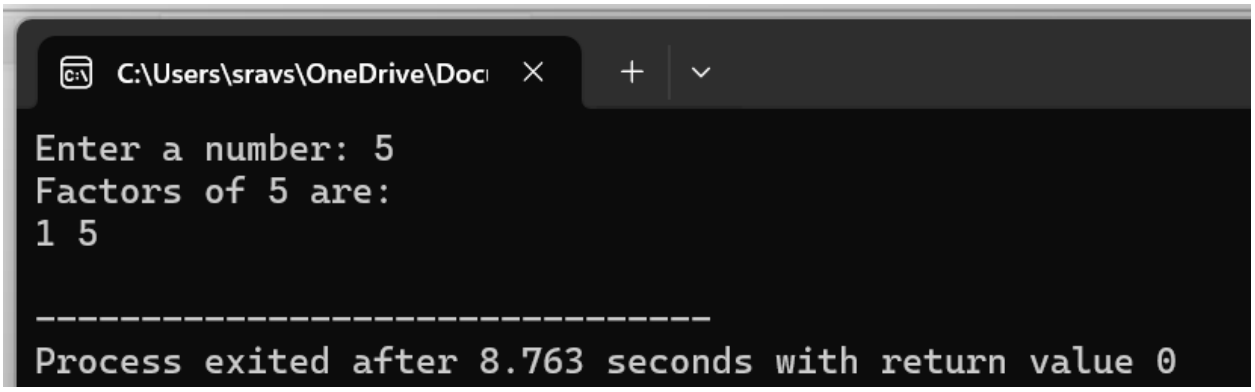
```
#include <stdio.h>
#include <stdlib.h>
#define MAX_ITEMS 100
#define MAX_BINS 100

int compare(const void *a, const void *b) {
return (*(int*)b - *(int*)a);
}
void containerLoader(int items[], int n, int binCapacity) {
int bins[MAX_BINS];
int binCount = 0;
int i, j;
for (i = 0; i < MAX_BINS; i++) {
bins[i] = 0;
}
qsort(items, n, sizeof(int), compare);
for (i = 0; i < n; i++) {
```

```
int item = items[i];
int placed = 0;
for (j = 0; j < binCount; j++) {
if (bins[j] + item <= binCapacity) {
bins[j] += item;
placed = 1;
break;
}
}
if (!placed) {
bins[binCount] = item;
binCount++;
}
}
printf("Number of bins used: %d\n", binCount);
for (i = 0; i < binCount; i++) {
printf("Bin %d: %d\n", i + 1, bins[i]);
}
}
int main() {
int items[MAX_ITEMS];
int n, binCapacity;
printf("Enter the number of items: ");
scanf("%d", &n);
printf("Enter the items:\n");
for (int i = 0; i < n; i++) {
scanf("%d", &items[i]);
}
printf("Enter the bin capacity: ");
scanf("%d", &binCapacity);
containerLoader(items, n, binCapacity);
return 0;
}
```



37. Generate the list of all factors for n value.

```
#include <stdio.h>
void printFactors(int n) {
printf("Factors of %d are:\n", n);
for (int i = 1; i <= n; i++) {
if (n % i == 0) {
printf("%d ", i);
}
}
printf("\n");
}
int main() {
int n;
printf("Enter a number: ");
scanf("%d", &n);
printFactors(n);
return 0;
}
```

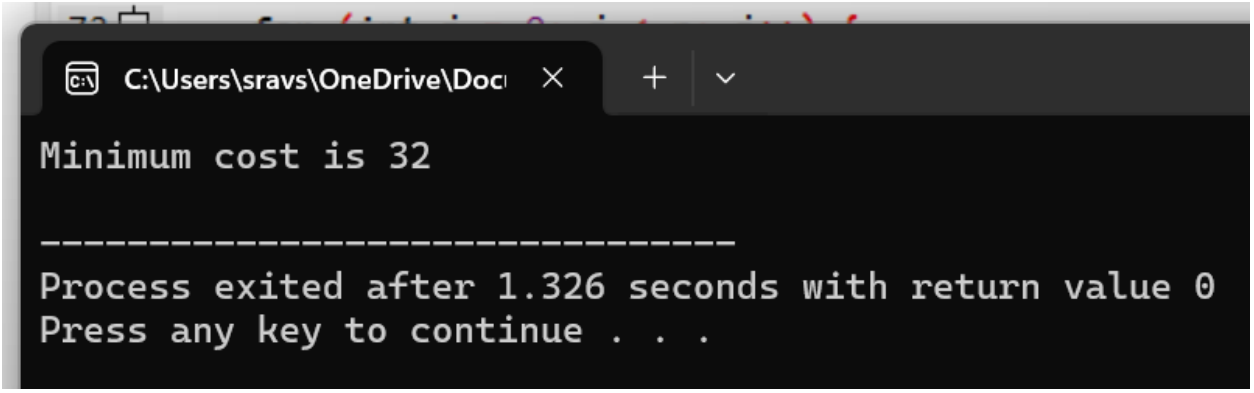


38. Assignment problem using branch and bound.

```
#include <stdio.h>
#include <limits.h>
#define N 4
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};
int visited[N] = {0};
int minCost = INT_MAX;
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];
          }
       }
```

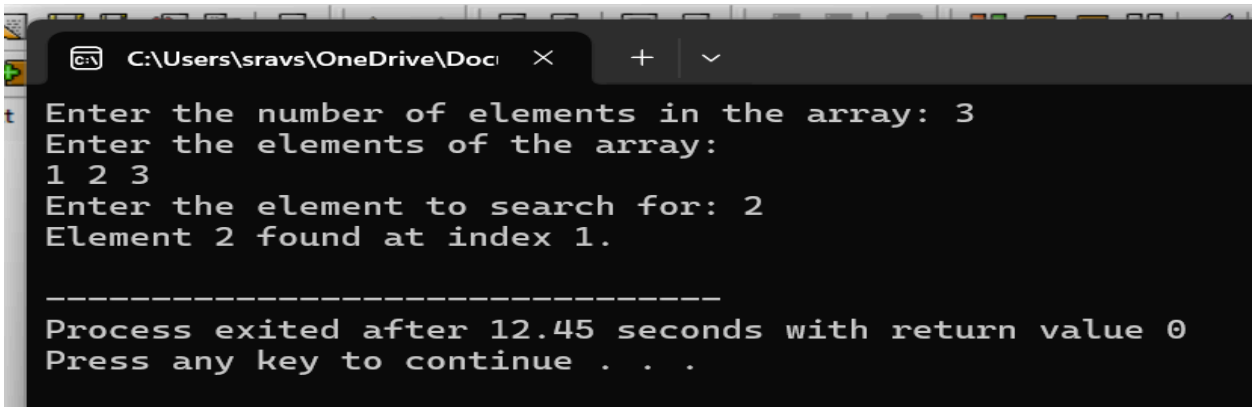
```
        bound += (min1 == INT_MAX) ? 0 : min1;
        bound += (min2 == INT_MAX) ? 0 : min2;
    }
    for (int j = 0; j < n; j++) {
        int min1 = INT_MAX, min2 = INT_MAX;
        for (int i = row; i < n; i++) {
            if (!visited[j] && costMatrix[i][j] < min1) {
                min2 = 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;
}
```



39. Linear 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;
}
```



40. Hamiltonian circuit Using backtracking method.

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

#define V 5
bool isSafe(int graph[V][V], int path[], int pos) {
    if (graph[path[pos-1]][path[pos]] == 0) {
        return false;
    }
```

```
    for (int i = 0; i < pos; i++) {
        if (path[i] == path[pos]) {
            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)) {
            path[pos] = v;
            if (hamCycleUtil(graph, path, pos + 1)) {
                return true;
            }
            path[pos] = -1;
        }
    }

    return false;
}
void findHamiltonianCircuit(int graph[V][V]) {
    int path[V];
    for (int i = 0; i < V; i++) {
        path[i] = -1;
    }
    path[0] = 0;

    if (hamCycleUtil(graph, path, 1) == false) {
        printf("No Hamiltonian Circuit found\n");
} else {
printf("Hamiltonian Circuit found:\n");
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, 1, 1, 1},
{1, 1, 0, 1, 1},
{1, 1, 1, 0, 1},
{0, 1, 1, 1, 0}
};
findHamiltonianCircuit(graph);
return 0;
}
```