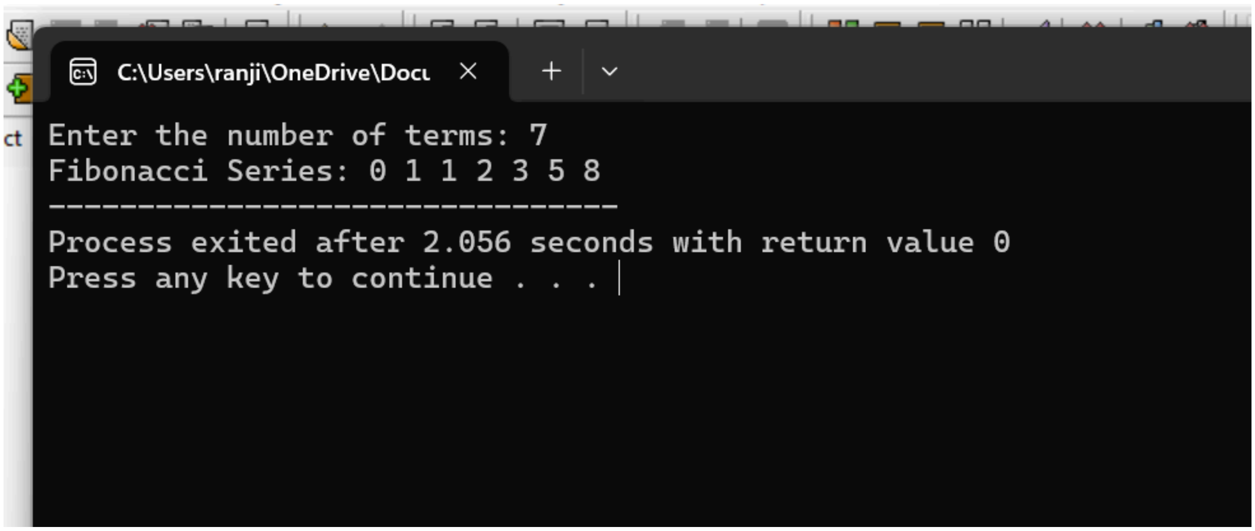
CSA0699 - DESIGN AND ANALYSIS OF ALGORITHM

DAY-1 (17/09/24)

1. Fibonacci series using recursion:

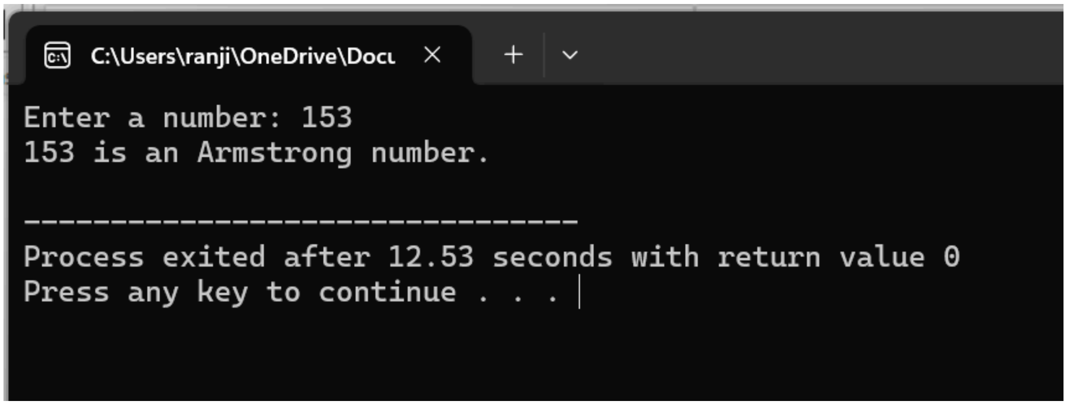
```
#include <stdio.h>
int fibonacci(int n) {
if (n <= 1)
return n;
else
return (fibonacci(n - 1) + fibonacci(n - 2));
}
int main() {
int n, i;
printf("Enter the number of terms: ");
scanf("%d", &n);
printf("Fibonacci Series: ");
for (i = 0; i < n; i++) {
printf("%d ", fibonacci(i));
}
return 0;
}
```



```
Enter the number of terms: 7
Fibonacci Series: 0 1 1 2 3 5 8
--------------------------------
Process exited after 2.056 seconds with return value 0
Press any key to continue . . .
```

2. Armstrong or not:

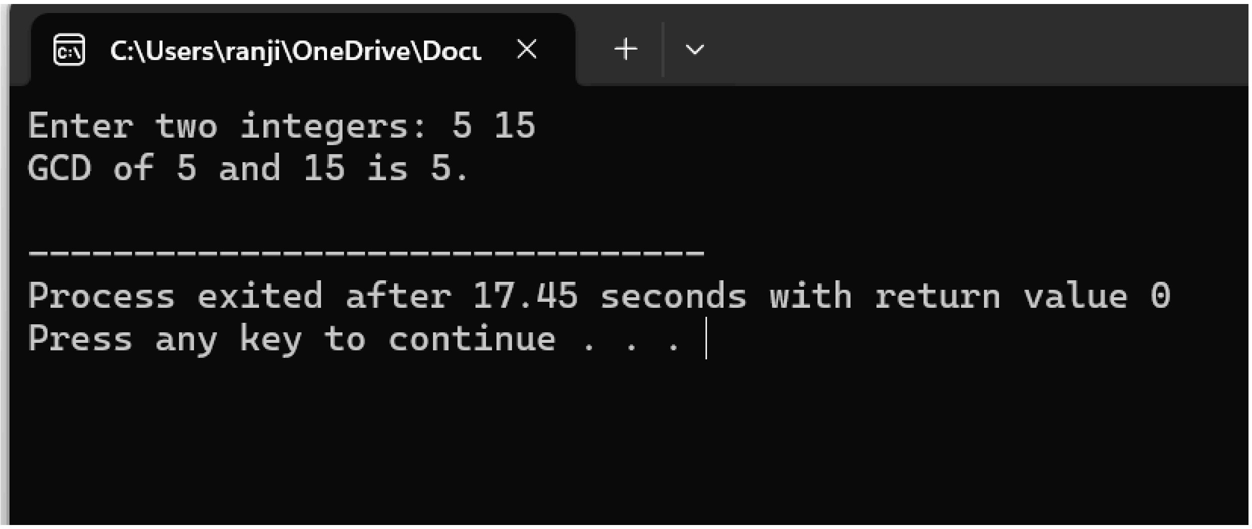
```
 #include <stdio.h>
#include <math.h>
int is_armstrong(int n) {
   int original = n, sum = 0, digits = 0, remainder;
   while (n != 0) {
      n /= 10;
      digits++;
   }
   n = original;
   while (n != 0) {
      remainder = n % 10;
      sum += pow(remainder, digits);
      n /= 10;
   }
   return (sum == original);
}
int main() {
   int number;
   printf("Enter a number: ");
   scanf("%d", &number);
   if (is_armstrong(number)) {
      printf("%d is an Armstrong number.\n", number);
   } else {
      printf("%d is not an Armstrong number.\n", number);
   }
   return 0;
}
```



```
Enter a number: 153
153 is an Armstrong number.

--------------------------------
Process exited after 12.53 seconds with return value 0
Press any key to continue . . .
```

3. GCD of two numbers:

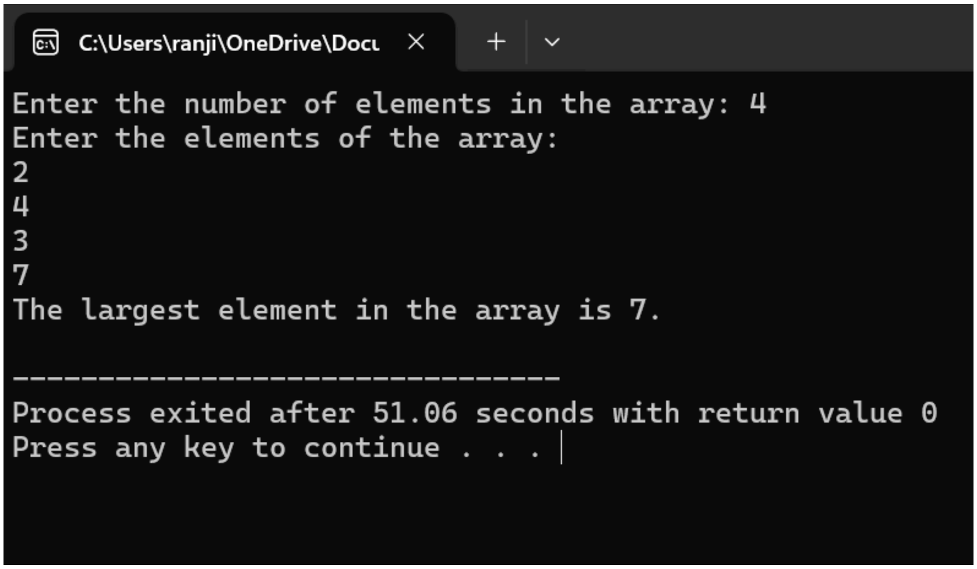
```
#include <stdio.h>
int gcd(int a, int b) {
if (b == 0)
return a;
else
return gcd(b, a % b);
}
int main() {
int num1, num2, result;
printf("Enter two integers: ");
scanf("%d %d", &num1, &num2);
result = gcd(num1, num2);
printf("GCD of %d and %d is %d.\n", num1, num2, result);
return 0;
}
```



```
Enter two integers: 5 15
GCD of 5 and 15 is 5.

--------------------------------
Process exited after 17.45 seconds with return value 0
Press any key to continue . . .
```

4. Largest Element of an Array:

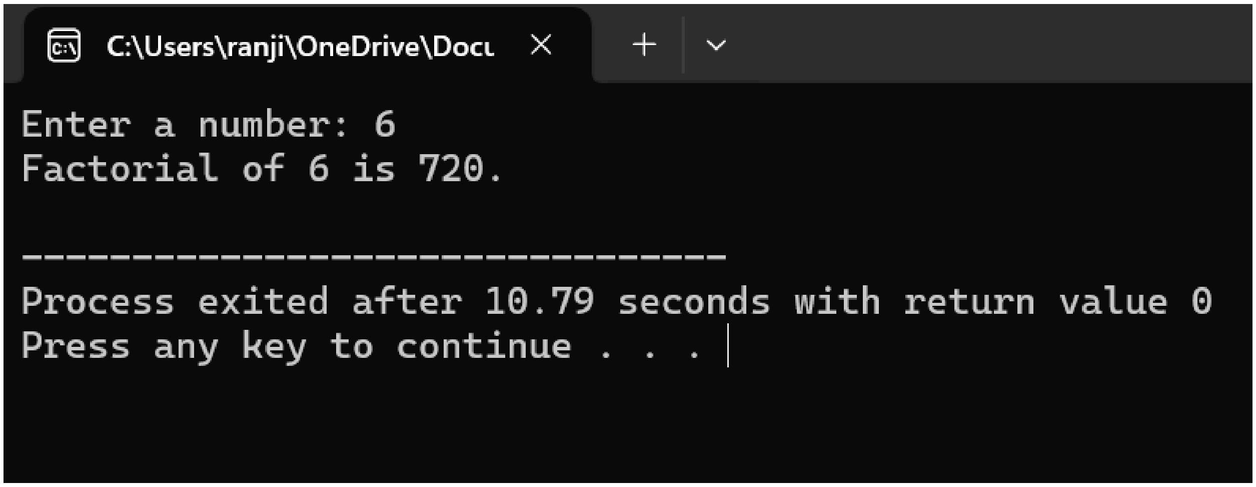
```
#include <stdio.h>
int main() {
int n, i;
int largest;
printf("Enter the number of elements in the array: ");
scanf("%d", &n);
int arr[n];
printf("Enter the elements of the array:\n");
for (i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
largest = arr[0];
for (i = 1; i < n; i++) {
if (arr[i] > largest)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



```
Enter the number of elements in the array: 4
Enter the elements of the array:
2
4
3
7
The largest element in the array is 7.

--------------------------------
Process exited after 51.06 seconds with return value 0
Press any key to continue . . .
```

5. Factorial of a number:

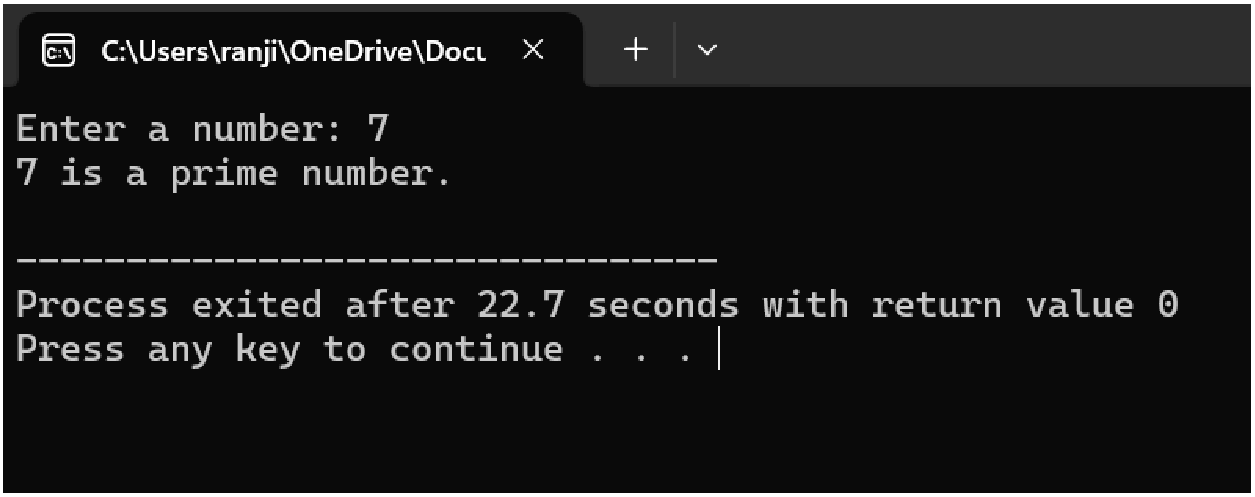
```
#include <stdio.h>
unsigned long long factorial(int n) {
unsigned long long fact = 1;
for (int i = 1; i <= n; ++i) {
fact *= i;
}
return fact;
}
int main() {
int num;
printf("Enter a number: ");
scanf("%d", &num);
if (num < 0) {
printf("Factorial is not defined for negative numbers.\n");
} else {
printf("Factorial of %d is %llu.\n", num, factorial(num));
}
return 0;
}
```



```
Enter a number: 6
Factorial of 6 is 720.

--------------------------------
Process exited after 10.79 seconds with return value 0
Press any key to continue . . .
```

6. Prime or not:

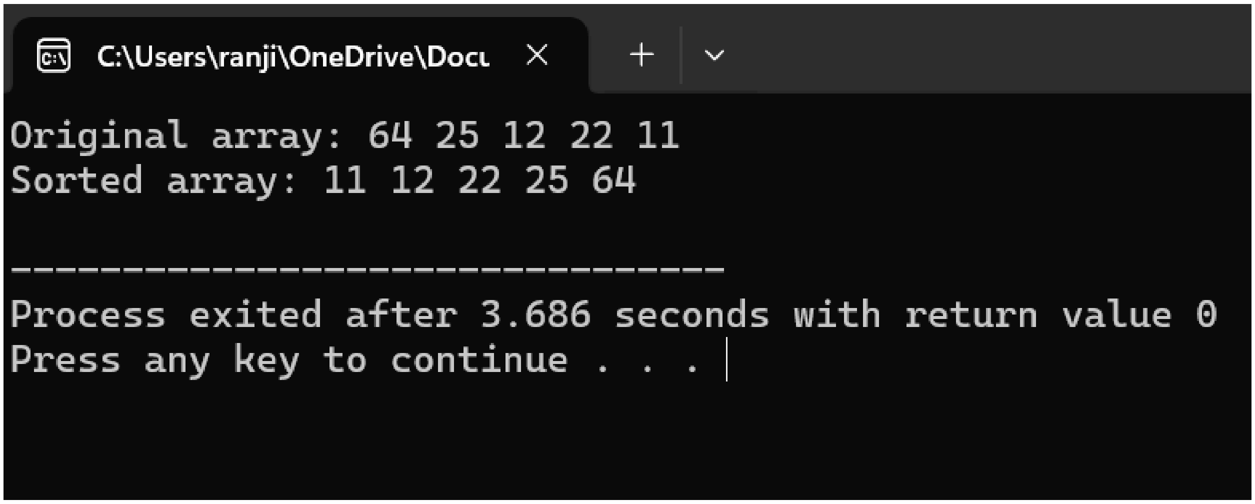
```
#include <stdio.h>
#include <stdbool.h>
#include <math.h>
bool isPrime(int num) {
if (num <= 1) {
return false;
}
for (int i = 2; i <= sqrt(num); i++) {
if (num % i == 0) {
return false;
}
}
return true;
}
int main() {
int num;
printf("Enter a number: ");
scanf("%d", &num);
if (isPrime(num)) {
printf("%d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



```
Enter a number: 7
7 is a prime number.

--------------------------------
Process exited after 22.7 seconds with return value 0
Press any key to continue . . .
```

7. Selection sort:

```
#include <stdio.h>
void selection_sort(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 print_array(int arr[], int size) {
    int i;
    for (i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Original array: ");
    print_array(arr, n);
    selection_sort(arr, n);
    printf("Sorted array: ");
    print_array(arr, n);
    return 0;
}
```

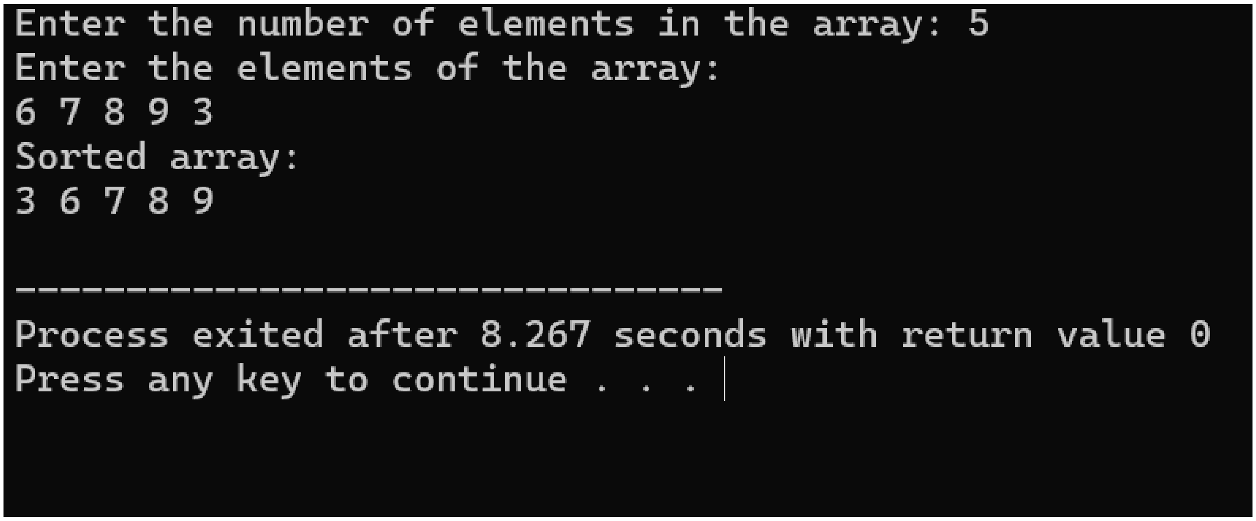


```
Original array: 64 25 12 22 11
Sorted array: 11 12 22 25 64

--------------------------------
Process exited after 3.686 seconds with return value 0
Press any key to continue . . .
```

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;
}
```



```
Enter the number of elements in the array: 5
Enter the elements of the array:
6 7 8 9 3
Sorted array:
3 6 7 8 9

--------------------------------
Process exited after 8.267 seconds with return value 0
Press any key to continue . . .
```

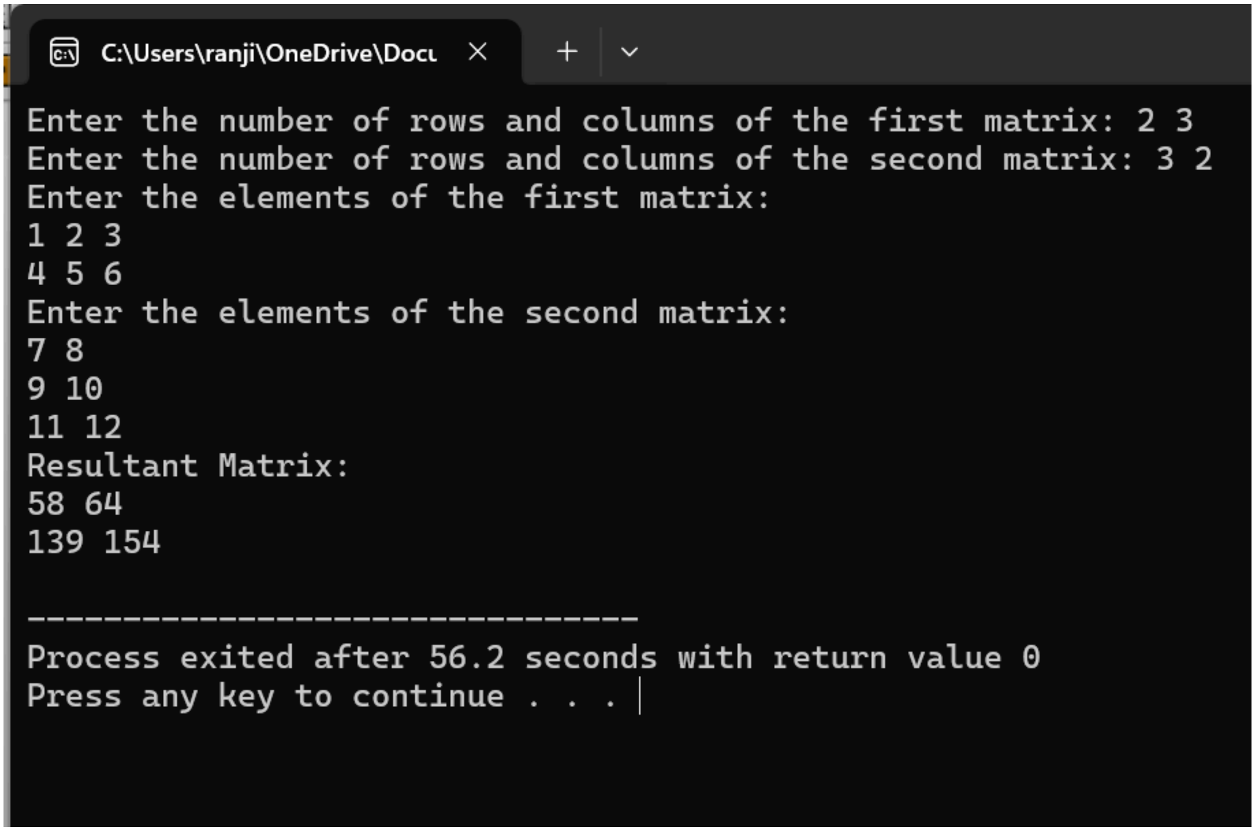
DAY-2(20/09/24)

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++) {
```

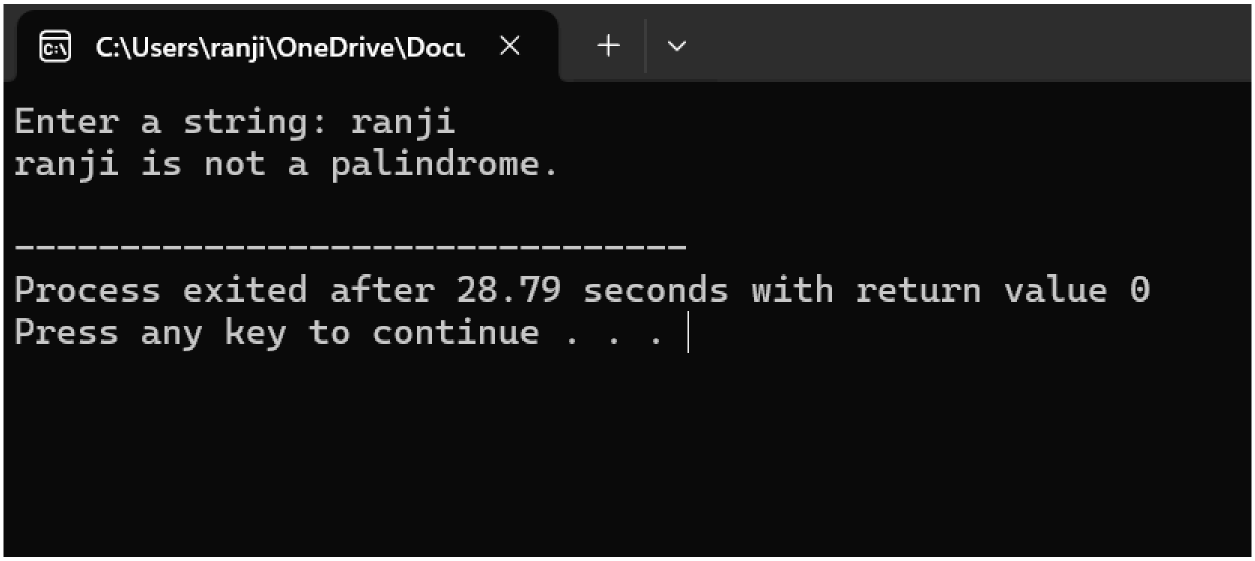
```
for (j = 0; j < c1; j++) {
scanf("%d", &firstMatrix[i][j]);
}
}
printf("Enter the elements of the second matrix:\n");
for (i = 0; i < r2; i++) {
for (j = 0; j < c2; j++) {
scanf("%d", &secondMatrix[i][j]);
}
}
multiplyMatrices(firstMatrix, secondMatrix, resultMatrix, r1, c1, r2,
c2);
printf("Resultant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



```
Enter the number of rows and columns of the first matrix: 2 3
Enter the number of rows and columns of the second matrix: 3 2
Enter the elements of the first matrix:
1 2 3
4 5 6
Enter the elements of the second matrix:
7 8
9 10
11 12
Resultant Matrix:
58 64
139 154

--------------------------------
Process exited after 56.2 seconds with return value 0
Press any key to continue . . .
```

10. Palindrome or not:

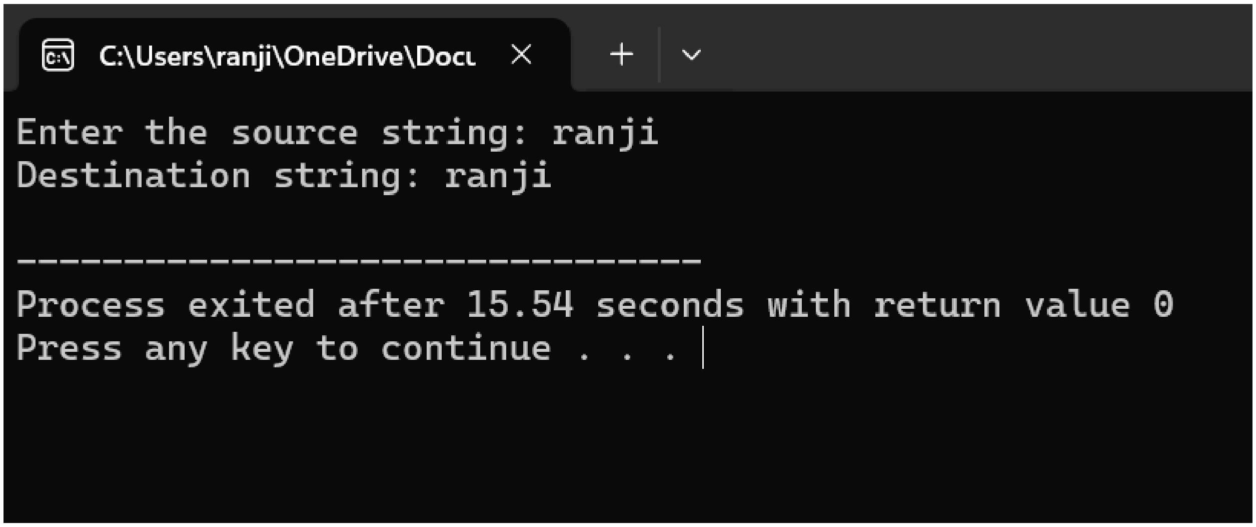
```
#include <stdio.h>
#include <string.h>
int isPalindrome(char str[]) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        if (str[i] != str[len - i - 1]) {
            return 0;
        }
    }
    return 1;
}
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);

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

    return 0;
}
```



```
Enter a string: ranji
ranji is not a palindrome.

--------------------------------
Process exited after 28.79 seconds with return value 0
Press any key to continue . . .
```

11. Copy one string to another:

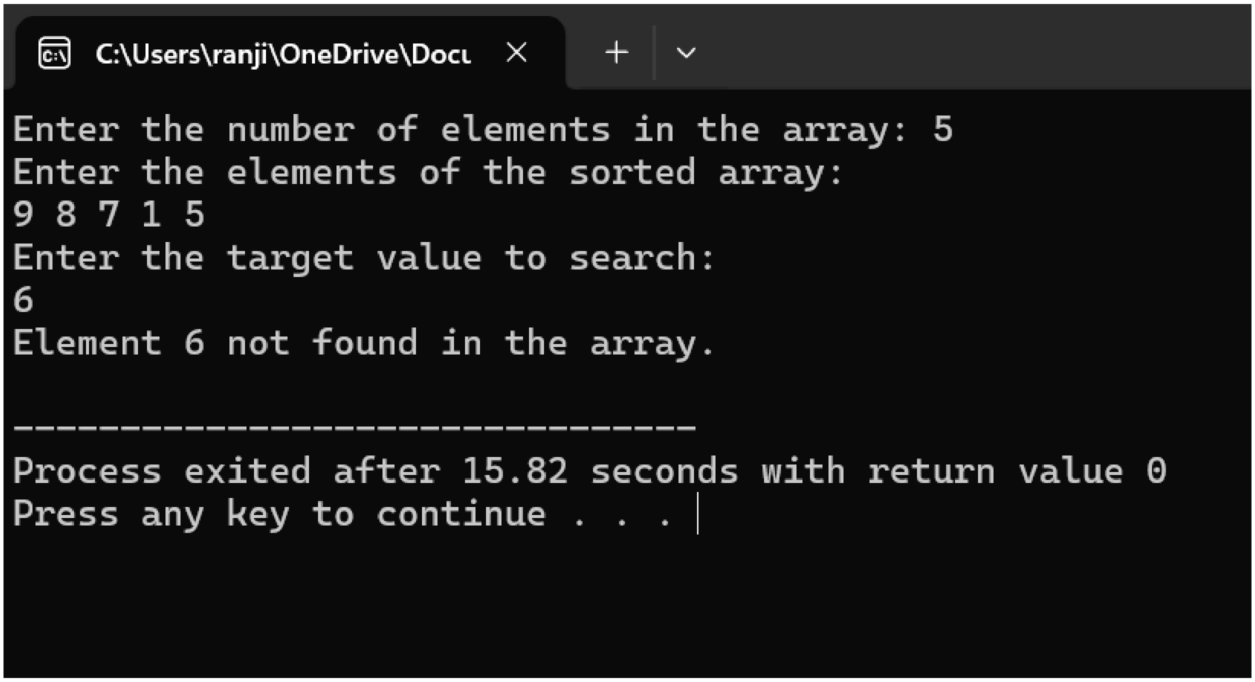
```
#include <stdio.h>
#include <string.h>
int main() {
 char source[100], destination[100];
 printf("Enter the source string: ");
 fgets(source, sizeof(source), stdin);
 source[strcspn(source, "\n")] = '\0';
 strcpy(destination, source);
 printf("Destination string: %s\n", destination);
 return 0;
}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



```
Enter the source string: ranji
Destination string: ranji

--------------------------------
Process exited after 15.54 seconds with return value 0
Press any key to continue . . .
```

12. Binary Search:

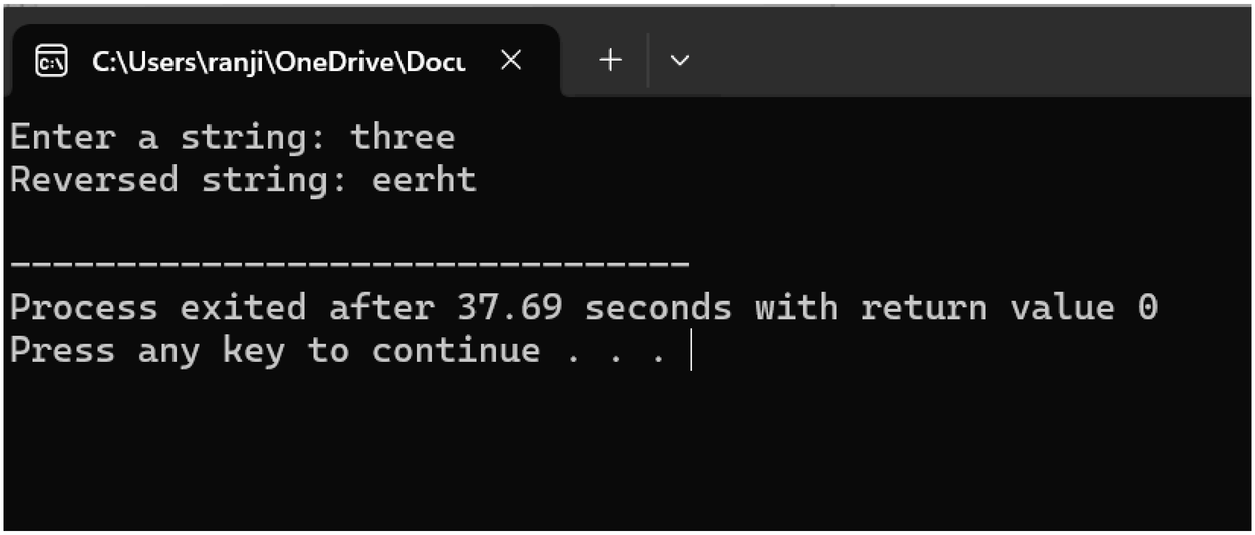
```
#include <stdio.h>
int binarySearch(int arr[], int size, int target) {
 int low = 0;
 int high = size - 1;
 int mid;
 while (low <= high) {
 mid = low + (high - low) / 2;
 if (arr[mid] == target) {
 return mid;
 }
 if (arr[mid] > target) {
 high = mid - 1;
 }
 else {
 low = mid + 1;
 }
 }
 return -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:\n ");
 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;
}
```



```
Enter the number of elements in the array: 5
Enter the elements of the sorted array:
9 8 7 1 5
Enter the target value to search:
6
Element 6 not found in the array.

--------------------------------
Process exited after 15.82 seconds with return value 0
Press any key to continue . . .
```

13. Reverse of String:

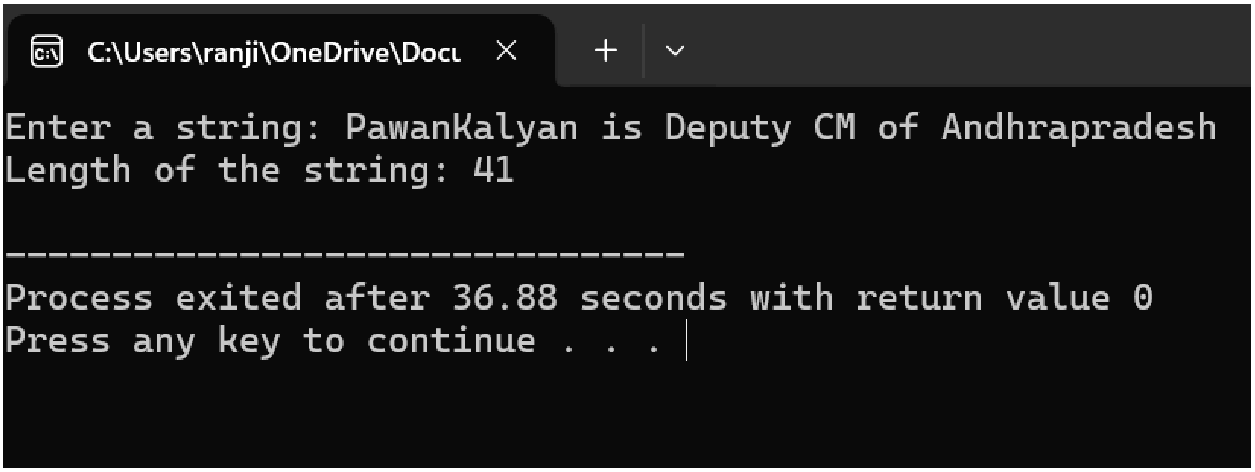
```
#include <stdio.h>
#include <string.h>
void printReverse(char str[]) {
 int length = strlen(str);
 for (int i = length - 1; i >= 0; i--) {
 printf("%c", str[i]);
 }
 printf("\n");
}
int main() {
 char str[100];
 printf("Enter a string: ");
 fgets(str, sizeof(str), stdin);
 str[strcspn(str, "\n")]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



```
Enter a string: three
Reversed string: eerht

--------------------------------
Process exited after 37.69 seconds with return value 0
Press any key to continue . . .
```

14. Length of String:

```
#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;
}
```



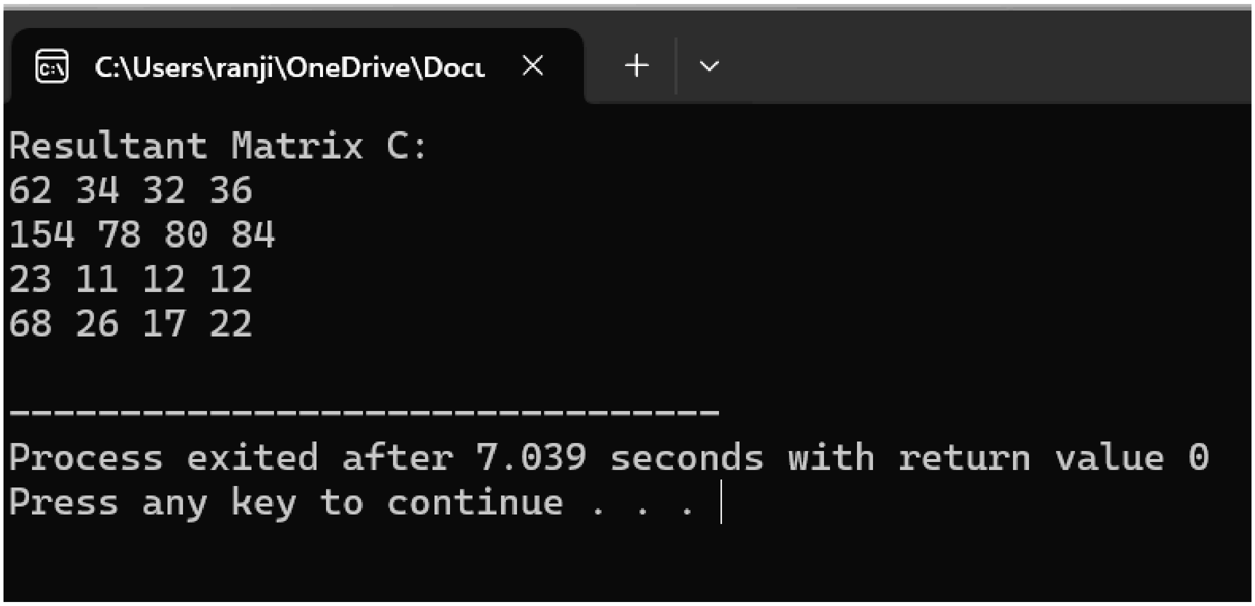
```
Enter a string: PawanKalyan is Deputy CM of Andhrapradesh
Length of the string: 41

--------------------------------
Process exited after 36.88 seconds with return value 0
Press any key to continue . . .
```

15. Strassens 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;
}
```

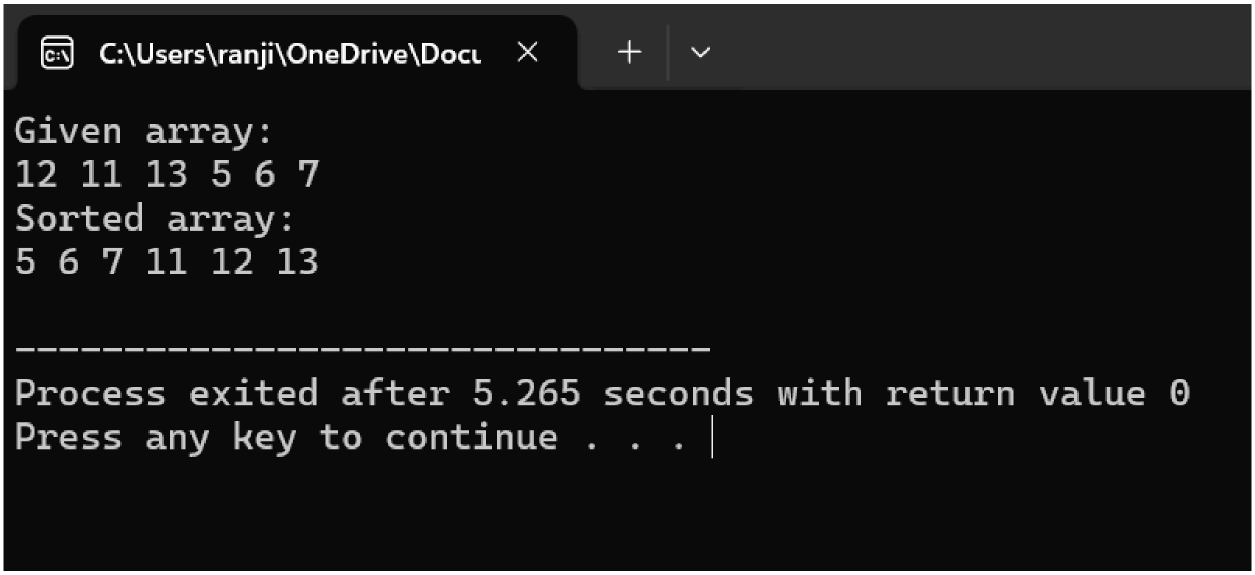


```
Resultant Matrix C:
62 34 32 36
154 78 80 84
23 11 12 12
68 26 17 22

--------------------------------
Process exited after 7.039 seconds with return value 0
Press any key to continue . . .
```

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;
}
```



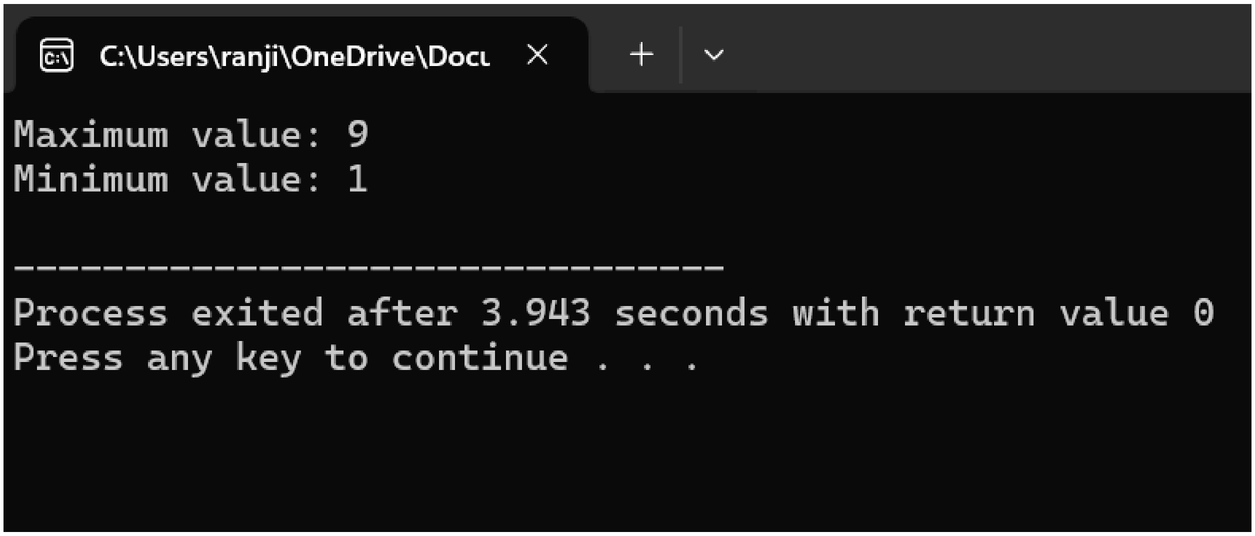
```
Given array:
12 11 13 5 6 7
Sorted array:
5 6 7 11 12 13

--------------------------------
Process exited after 5.265 seconds with return value 0
Press any key to continue . . .
```

DAY-3(19-09-24)

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];
```

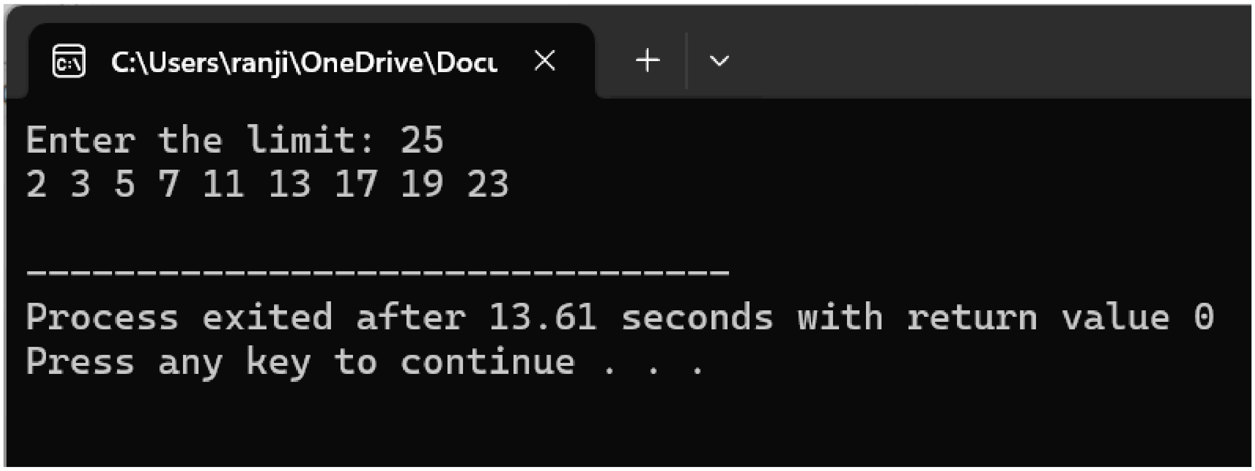
```
            *min = arr[left];
        }
    } else {
        int mid = left + (right - left) / 2;
        int leftMax, leftMin, rightMax, rightMin;

        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;
}
```



```
Maximum value: 9
Minimum value: 1

--------------------------------
Process exited after 3.943 seconds with return value 0
Press any key to continue . . .
```

18. Generate all the prime numbers:

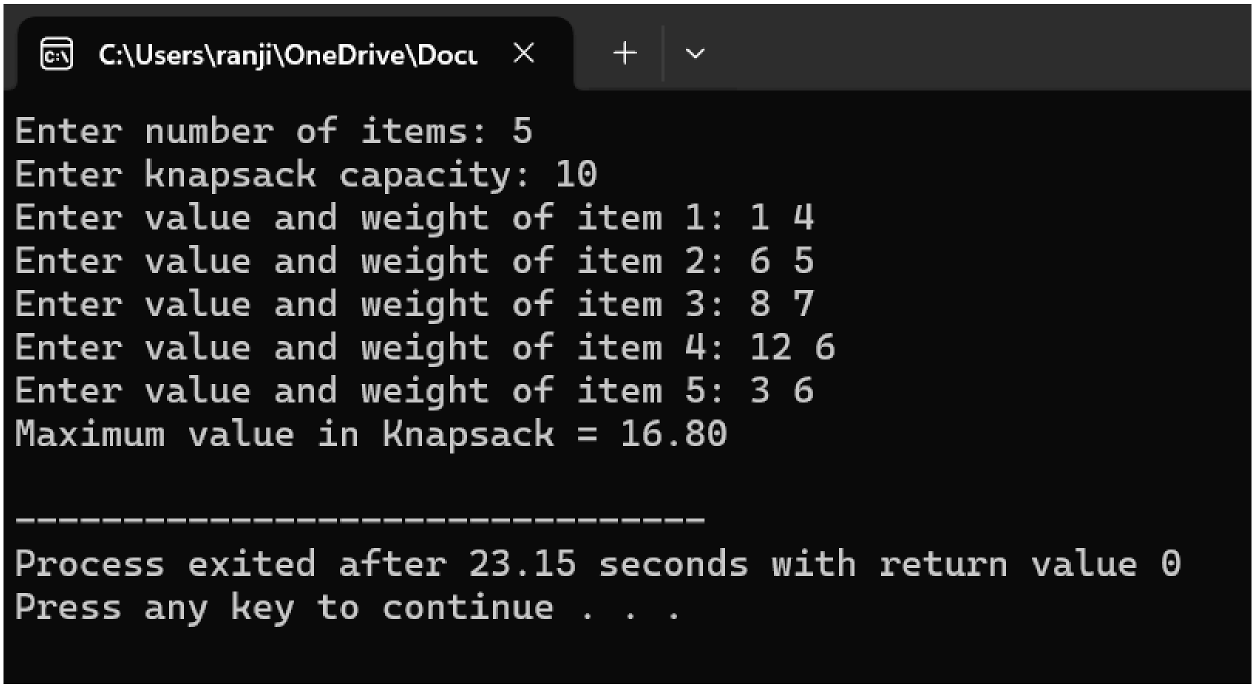
```
#include <stdio.h>
#include <stdbool.h>
void generatePrimes(int limit) {
   bool prime[limit + 1];
   for (int i = 0; i <= limit; i++)
      prime[i] = true;
   for (int p = 2; p * p <= limit; p++) {
      if (prime[p]) {
         for (int i = p * p; i <= limit; i += p)
            prime[i] = false;
      }
   }
   for (int p = 2; p <= limit; p++) {
      if (prime[p])
         printf("%d ", p);
   }
   printf("\n");
}
int main() {
   int limit;
   printf("Enter the limit: ");
   scanf("%d", &limit);
   generatePrimes(limit);
   return 0;
}
```



```
Enter the limit: 25
2 3 5 7 11 13 17 19 23

--------------------------------
Process exited after 13.61 seconds with return value 0
Press any key to continue . . .
```

19. Knapsack problem using greedy techniques:

```
#include <stdio.h>
struct Item {
    int value, weight;
};
void sortItems(struct Item items[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            double r1 = (double)items[j].value / items[j].weight;
            double r2 = (double)items[j + 1].value / items[j + 1].weight;
            if (r1 < r2) {
                struct Item temp = items[j];
                items[j] = items[j + 1];
                items[j + 1] = temp;
            }
        }
    }
}
double knapsack(struct Item items[], int n, int W) {
    sortItems(items, n);
    double maxValue = 0.0;
    for (int i = 0; i < n; i++) {
        if (W == 0)
            break;
        if (items[i].weight <= W) {
            maxValue += items[i].value;
            W -= items[i].weight;
        } else {
            maxValue += items[i].value * ((double)W / items[i].weight);
            W = 0;
        }
    }

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

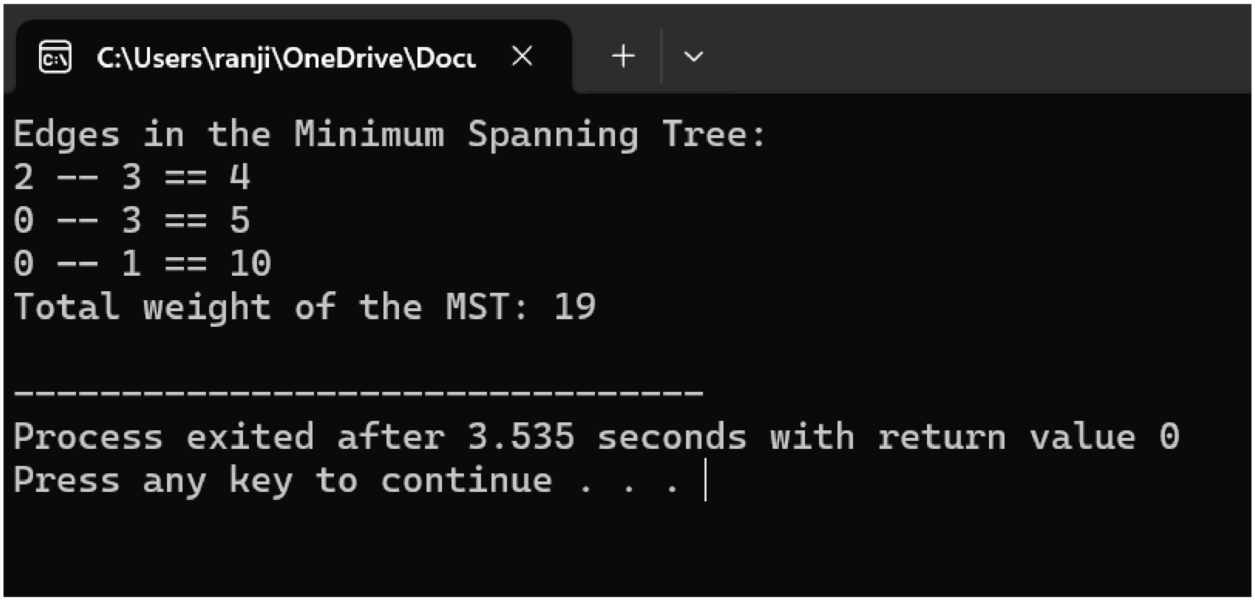


```
Enter number of items: 5
Enter knapsack capacity: 10
Enter value and weight of item 1: 1 4
Enter value and weight of item 2: 6 5
Enter value and weight of item 3: 8 7
Enter value and weight of item 4: 12 6
Enter value and weight of item 5: 3 6
Maximum value in Knapsack = 16.80

--------------------------------
Process exited after 23.15 seconds with return value 0
Press any key to continue . . .
```

20. MST using greedy techniques:

```
#include <stdio.h>
#include <stdlib.h>
struct Edge {
   int src, dest, weight;
};
struct Subset {
   int parent;
   int rank;
};
int find(struct Subset subsets[], int i) {
   if (subsets[i].parent != i)
      subsets[i].parent = find(subsets, subsets[i].parent);
   return subsets[i].parent;
}
void unionSets(struct Subset subsets[], int x, int y) {
   int rootX = find(subsets, x);
   int rootY = find(subsets, y);
```

```
    if (subsets[rootX].rank < subsets[rootY].rank)
        subsets[rootX].parent = rootY;
    else if (subsets[rootX].rank > subsets[rootY].rank)
        subsets[rootY].parent = rootX;
    else {
        subsets[rootY].parent = rootX;
        subsets[rootX].rank++;
    }
}
int compareEdges(const void* a, const void* b) {
    struct Edge* edgeA = (struct Edge*)a;
    struct Edge* edgeB = (struct Edge*)b;
    return edgeA->weight > edgeB->weight;
}
void kruskalMST(struct Edge edges[], int V, int E) {
    qsort(edges, E, sizeof(edges[0]), compareEdges);
    struct Edge result[V];
    struct Subset* subsets = (struct Subset*) malloc(V * sizeof(struct Subset));
    for (int v = 0; v < V; v++) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }
    int e = 0;
    int i = 0;
    while (e < V - 1 && i < E) {
        struct Edge nextEdge = edges[i++];
        int x = find(subsets, nextEdge.src);
        int y = find(subsets, nextEdge.dest);
        if (x != y) {
            result[e++] = nextEdge;
            unionSets(subsets, x, y);
        }
    }
    printf("Edges in the Minimum Spanning Tree:\n");
    int totalWeight = 0;
    for (i = 0; i < e; i++) {
        printf("%d -- %d == %d\n", result[i].src, result[i].dest, result[i].weight);
        totalWeight += result[i].weight;
    }
    printf("Total weight of the MST: %d\n", totalWeight);
    free(subsets);
}
int main() {
    int V = 4;
    int E = 5;
    struct Edge edges[] = {
        {0, 1, 10},
        {0, 2, 6},
        {0, 3, 5},
        {1, 3, 15},
        {2, 3, 4}
    };
    kruskalMST(edges, V, E);
    return 0;
}
```

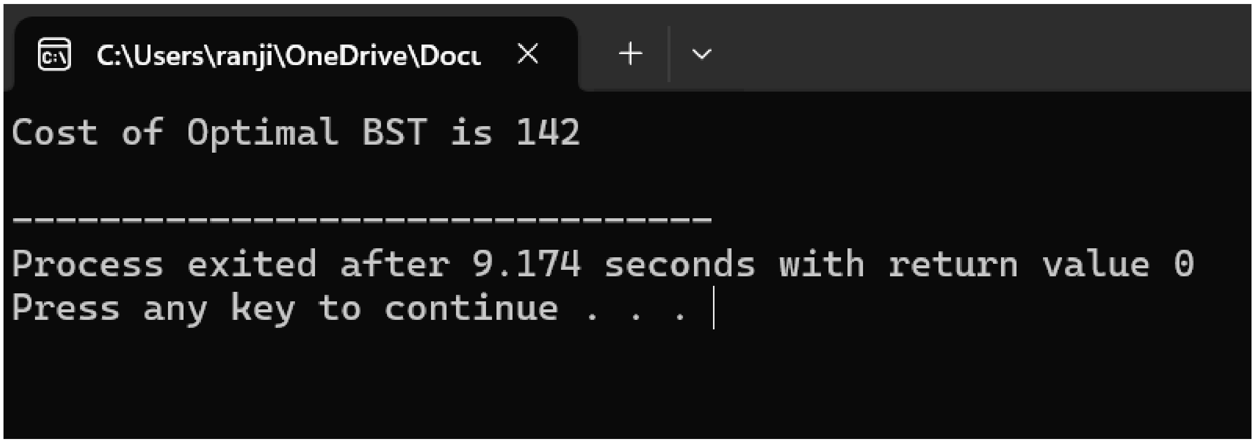


```
Edges in the Minimum Spanning Tree:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Total weight of the MST: 19

--------------------------------
Process exited after 3.535 seconds with return value 0
Press any key to continue . . .
```

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

```
#include <stdio.h>
#include <limits.h>
int sum(int freq[], int i, int j) {
    int s = 0;
    for (int k = i; k <= j; k++)
        s += freq[k];
    return s;
}
int optimalBST(int keys[], int freq[], int n) {
    int cost[n][n];
    for (int i = 0; i < n; i++)
        cost[i][i] = freq[i];
    for (int L = 2; L <= n; L++) {
        for (int i = 0; i <= n - L; i++) {
            int j = i + L - 1;
            cost[i][j] = INT_MAX;
                int c = ((r > i) ? cost[i][r - 1] : 0) +
                        ((r < j) ? cost[r + 1][j] : 0) +
                        sum(freq, i, j);

                if (c < cost[i][j])
                    cost[i][j] = c;
            }
        }
    }
    return cost[0][n - 1];
}
int main() {
    int keys[] = {10, 12, 20};
    int freq[] = {34, 8, 50};
    int n = sizeof(keys) / sizeof(keys[0]);
    printf("Cost of Optimal BST is %d\n", optimalBST(keys, freq, n));
    return 0;
```

```
}
```

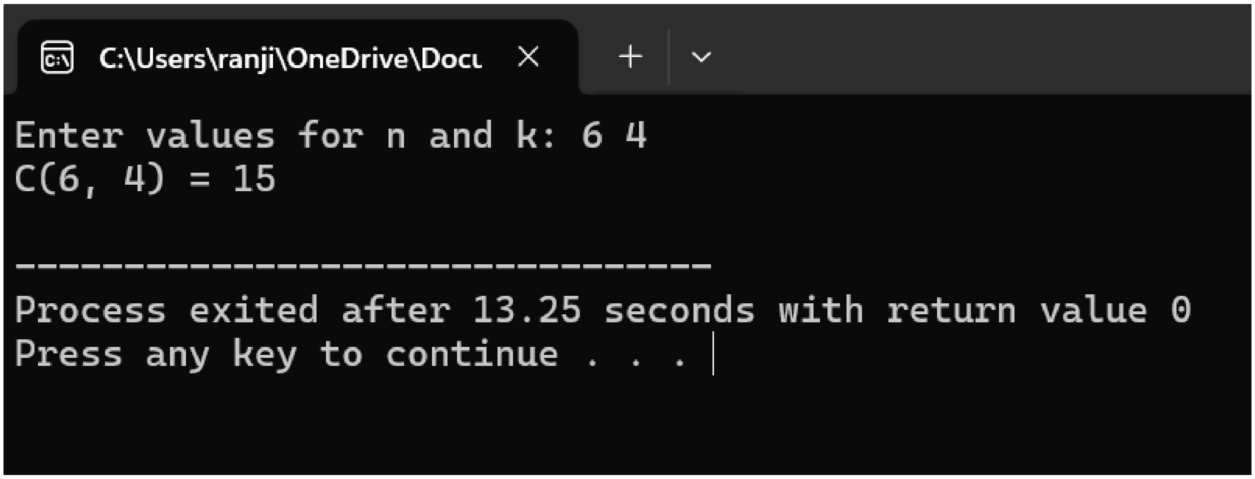


```
Cost of Optimal BST is 142

--------------------------------
Process exited after 9.174 seconds with return value 0
Press any key to continue . . .
```

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

```
#include <stdio.h>
int binomialCoefficient(int n, int k) {
   int dp[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) {
            dp[i][j] = 1;
         } else {
            dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];
         }
      }
   }
   return dp[n][k];
}
int main() {
   int n, k;
   printf("Enter values for n and k: ");
   scanf("%d %d", &n, &k);
   if (k > n) {
      printf("Invalid input: k cannot be greater than n.\n");
```

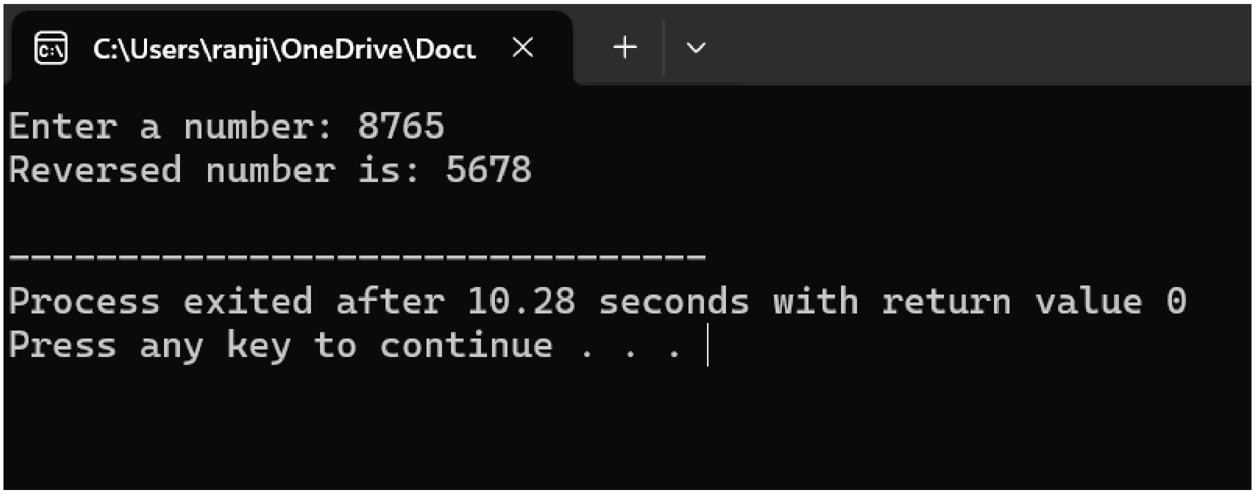
```
        return 1;
    }
    printf("C(%d, %d) = %d\n", n, k, binomialCoefficient(n, k));
    return 0;
}
```



```
Enter values for n and k: 6 4
C(6, 4) = 15

--------------------------------
Process exited after 13.25 seconds with return value 0
Press any key to continue . . .
```

23. Reverse of a given number:

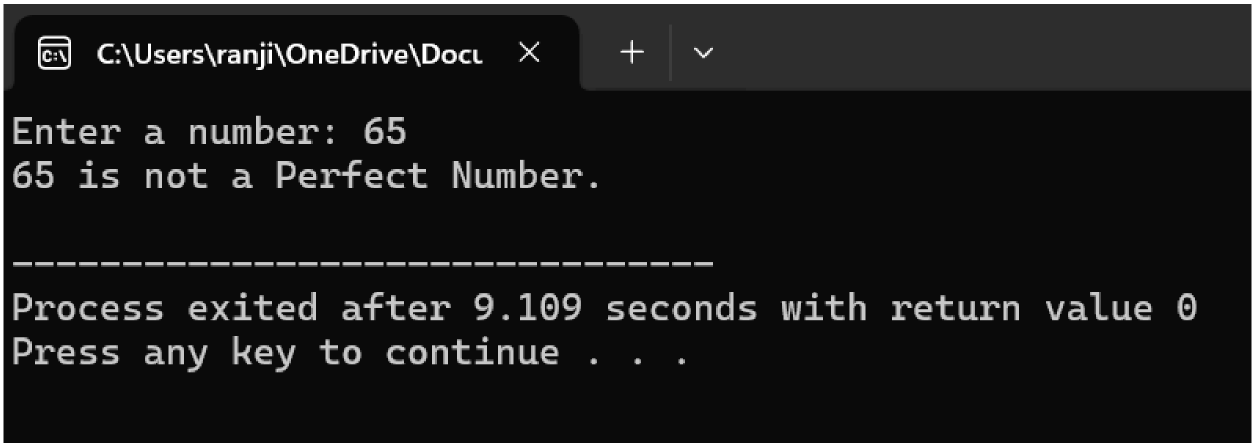
```
#include <stdio.h>
int reverseNumber(int num) {
    int reversed = 0;
    while (num != 0) {
        int digit = num % 10;
        reversed = reversed * 10 + digit;
        num /= 10;
    }
    return reversed;
}
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("Reversed number is: %d\n", reverseNumber(num));
    return 0;
}
```



```
Enter a number: 8765
Reversed number is: 5678

--------------------------------
Process exited after 10.28 seconds with return value 0
Press any key to continue . . .
```

24. 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 num;
   printf("Enter a number: ");
   scanf("%d", &num);
   if (isPerfect(num))
      printf("%d is a Perfect Number.\n", num);
   else
      printf("%d is not a Perfect Number.\n", num);

   return 0;
}
```



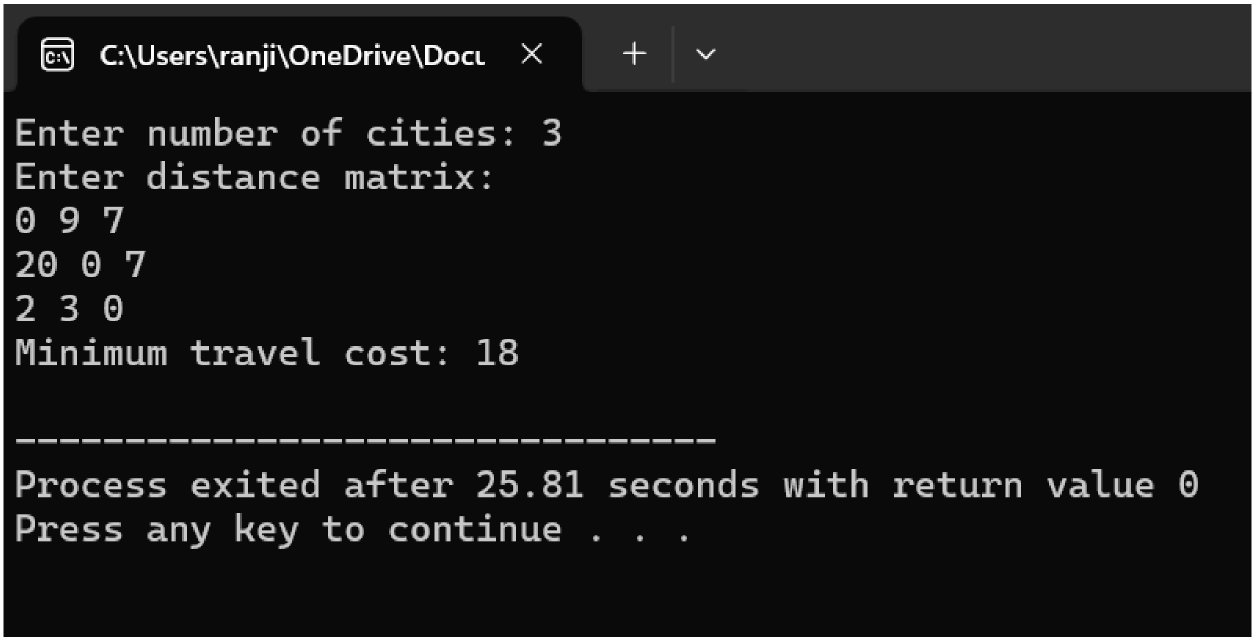
```
Enter a number: 65
65 is not a Perfect Number.

--------------------------------
Process exited after 9.109 seconds with return value 0
Press any key to continue . . .
```

DAY-4(20-09-24)

25. Traveling salesman problem using dynamic programming:

```
#include <stdio.h>
#define INF 9999999
int n;
int dist[10][10];
int dp[1024][10];
int tsp(int mask, int pos) {
   if (mask == ((1 << n) - 1))
      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 = (newAns < ans) ? newAns : ans;
      }
```

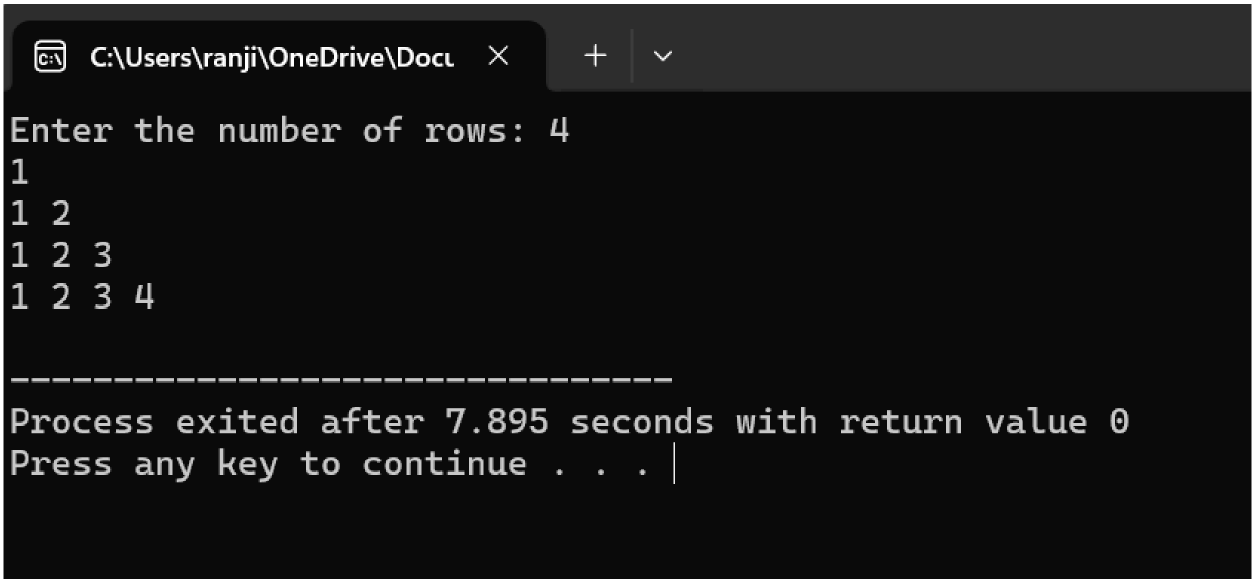
```
    }
    return dp[mask][pos] = ans;
}
int main() {
    printf("Enter number of cities: ");
    scanf("%d", &n);
    printf("Enter distance matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &dist[i][j]);
    for (int i = 0; i < (1 << n); i++)
        for (int j = 0; j < n; j++)
            dp[i][j] = -1;
    printf("Minimum travel cost: %d\n", tsp(1, 0));
    return 0;
}
```



```
Enter number of cities: 3
Enter distance matrix:
0 9 7
20 0 7
2 3 0
Minimum travel cost: 18

--------------------------------
Process exited after 25.81 seconds with return value 0
Press any key to continue . . .
```

26. Right angled triangle Format:

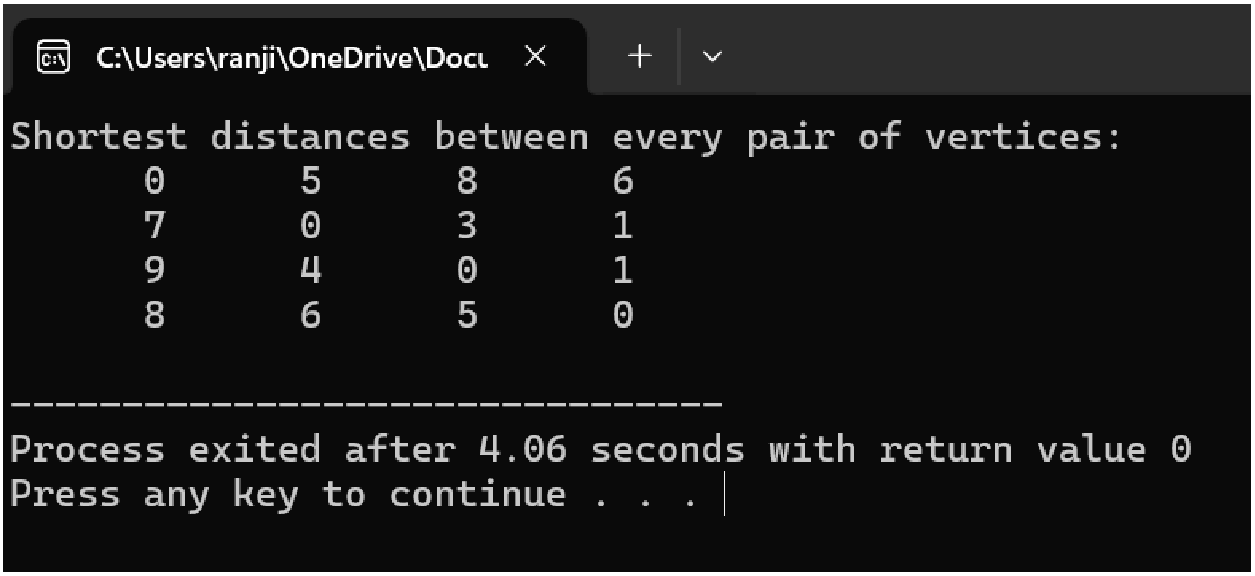
```
#include <stdio.h>
int main() {
    int n;
    printf("Enter the number of rows: ");
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```



```
Enter the number of rows: 4
1
1 2
1 2 3
1 2 3 4

--------------------------------
Process exited after 7.895 seconds with return value 0
Press any key to continue . . .
```

27. Floyd's algorithm:

```
#include <stdio.h>
#define INF 99999
#define V 4
void printSolution(int dist[][V]);
void floydWarshall(int graph[][V]) {
    int dist[V][V], i, j, k;
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];
    for (k = 0; k < V; k++) {
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    printSolution(dist);
}
void printSolution(int dist[][V]) {
    printf("Shortest distances between every pair of vertices:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INF)
                printf("%7s", "INF");
            else
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
}
int main() {
    int graph[V][V] = {
        {0, 5, INF, 10},
        {7, 0, 3, 1},
        {9, 4, 0, 1},
        {8, 6, 5, 0}
    };
    floydWarshall(graph);
    return 0;
}
```

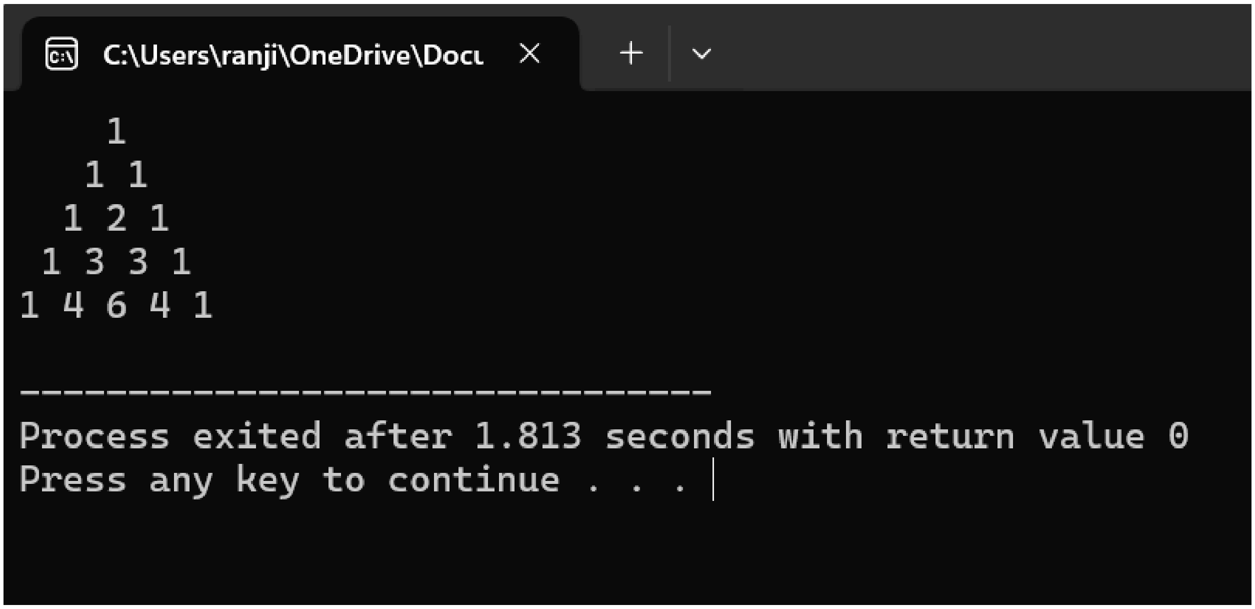


```
Shortest distances between every pair of vertices:
      0       5       8       6
      7       0       3       1
      9       4       0       1
      8       6       5       0

--------------------------------
Process exited after 4.06 seconds with return value 0
Press any key to continue . . .
```

28. Pascal triangle:

```
#include <stdio.h>
#define N 5
void printPascalTriangle(int n) {
   int arr[n][n];
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         arr[i][j] = 0;
      }
   }
   for (int i = 0; i < n; i++) {
      arr[i][0] = 1;
      arr[i][i] = 1;
      for (int j = 1; j < i; j++) {
         arr[i][j] = arr[i-1][j-1] + arr[i-1][j];
      }
   }
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n-i-1; j++) {
         printf(" ");
      }
```

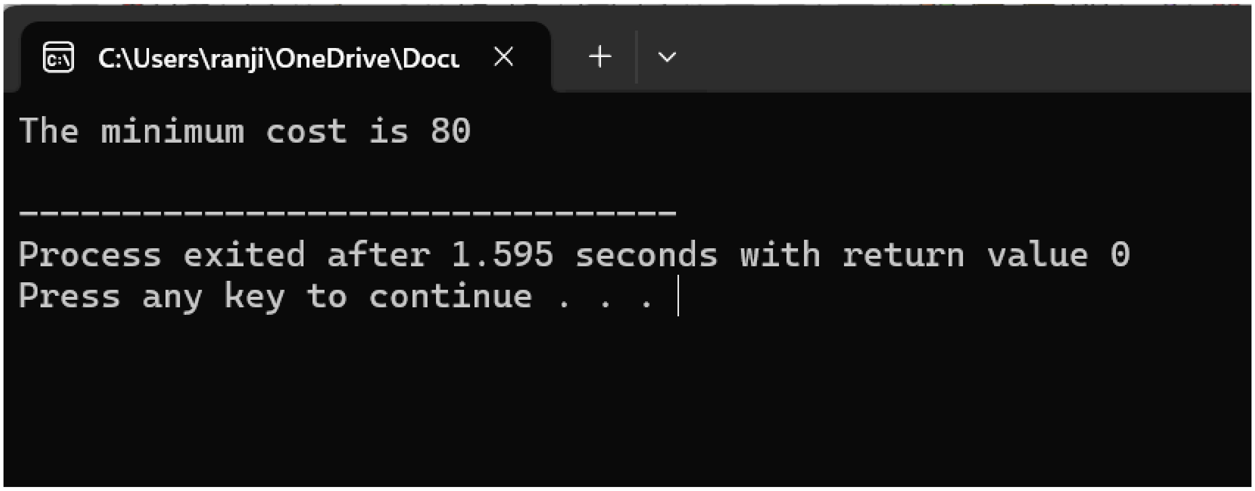
```
        for (int j = 0; j <= i; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
int main() {
    printPascalTriangle(N);
    return 0;
}
```



```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1

--------------------------------
Process exited after 1.813 seconds with return value 0
Press any key to continue . . .
```

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

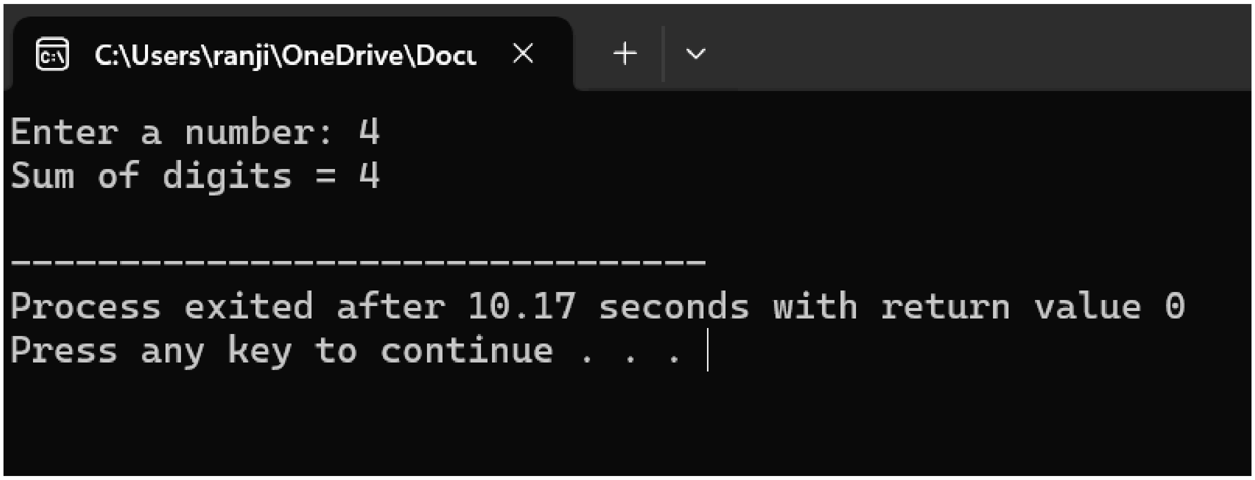
```
#include <stdio.h>
#include <limits.h>
#define V 4
int tsp(int graph[][V], int dp[][1 << V], int pos, int visited) {
    if (visited == (1 << V) - 1) {
        return graph[pos][0];
    }
    if (dp[pos][visited] != -1) {
        return dp[pos][visited];
    }
    int ans = INT_MAX;
    for (int city = 0; city < V; city++) {
        if ((visited & (1 << city)) == 0) {
            int newAns = graph[pos][city] + tsp(graph, dp, city, visited | (1 << city));
            ans = (ans < newAns) ? ans : newAns;
        }
    }
    return dp[pos][visited] = ans;
}
int main() {
    int graph[V][V] = {
        {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}
    };

    int dp[V][1 << V];
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < (1 << V); j++) {
            dp[i][j] = -1;
        }
    }
    printf("The minimum cost is %d\n", tsp(graph, dp, 0, 1));
    return 0;
}
```



```
The minimum cost is 80

--------------------------------
Process exited after 1.595 seconds with return value 0
Press any key to continue . . .
```

30. Sum of digits:

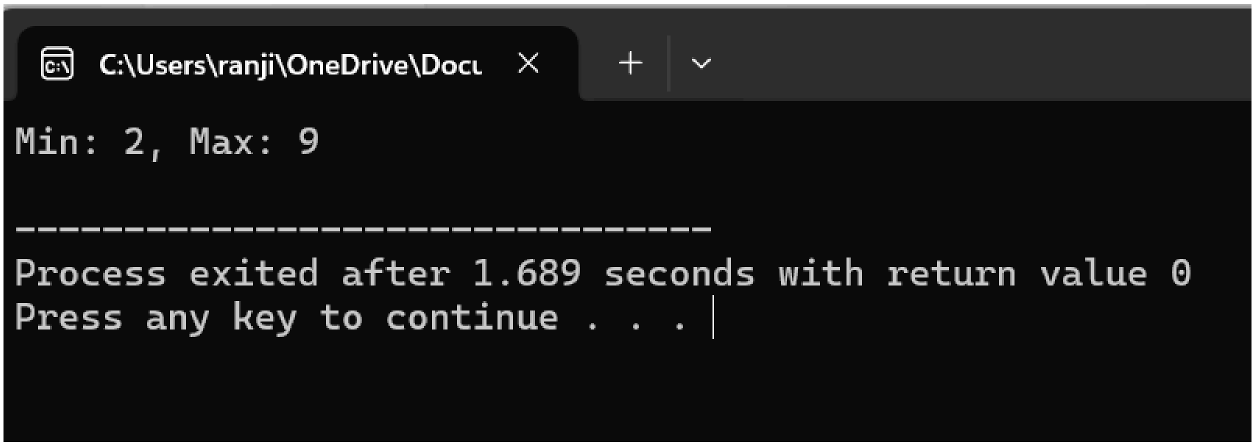
```
#include <stdio.h>
int sumOfDigits(int num) {
    int sum = 0;

    while (num != 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("Sum of digits = %d\n", sumOfDigits(num));
    return 0;
}
```



```
Enter a number: 4
Sum of digits = 4

--------------------------------
Process exited after 10.17 seconds with return value 0
Press any key to continue . . .
```

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

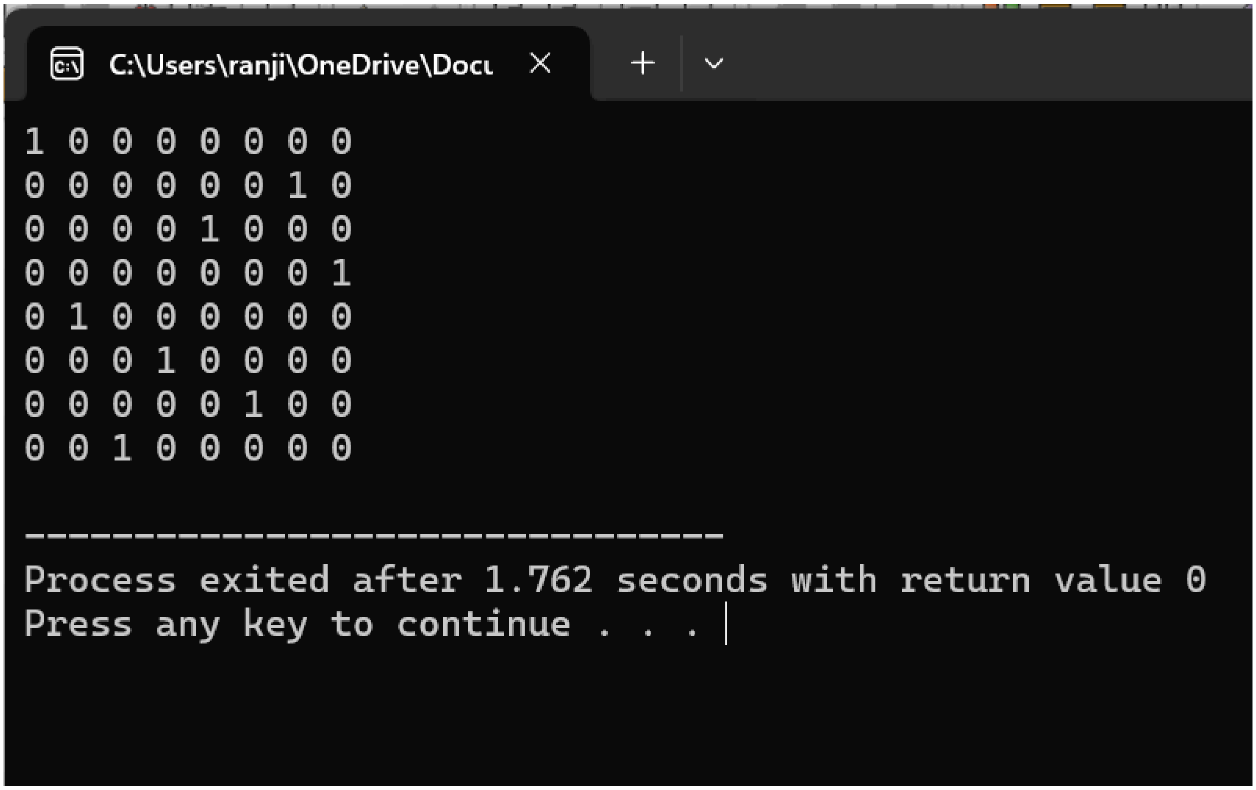
```
#include <stdio.h>
int main() {
   int arr[] = {3, 7, 2, 9, 4};
   int n = sizeof(arr) / sizeof(arr[0]);
   int min = arr[0], max = arr[0];

   for (int i = 1; i < n; i++) {
      if (arr[i] < min) min = arr[i];
      if (arr[i] > max) max = arr[i];
   }
   printf("Min: %d, Max: %d\n", min, max);
   return 0;
}
```



```
Min: 2, Max: 9

--------------------------------
Process exited after 1.689 seconds with return value 0
Press any key to continue . . .
```

32. N- Queen problem using Backtracking:

```
#include <stdio.h>
#include <stdbool.h>
#define N 8
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; i < N && j >= 0; 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;
}
bool solveNQ() {
    int board[N][N] = {0};

    if (solveNQUtil(board, 0)) {
        printSolution(board);
        return true;
    } else {
        printf("Solution does not exist.\n");
        return false;
    }
}
int main() {
    solveNQ();
    return 0;
}
```

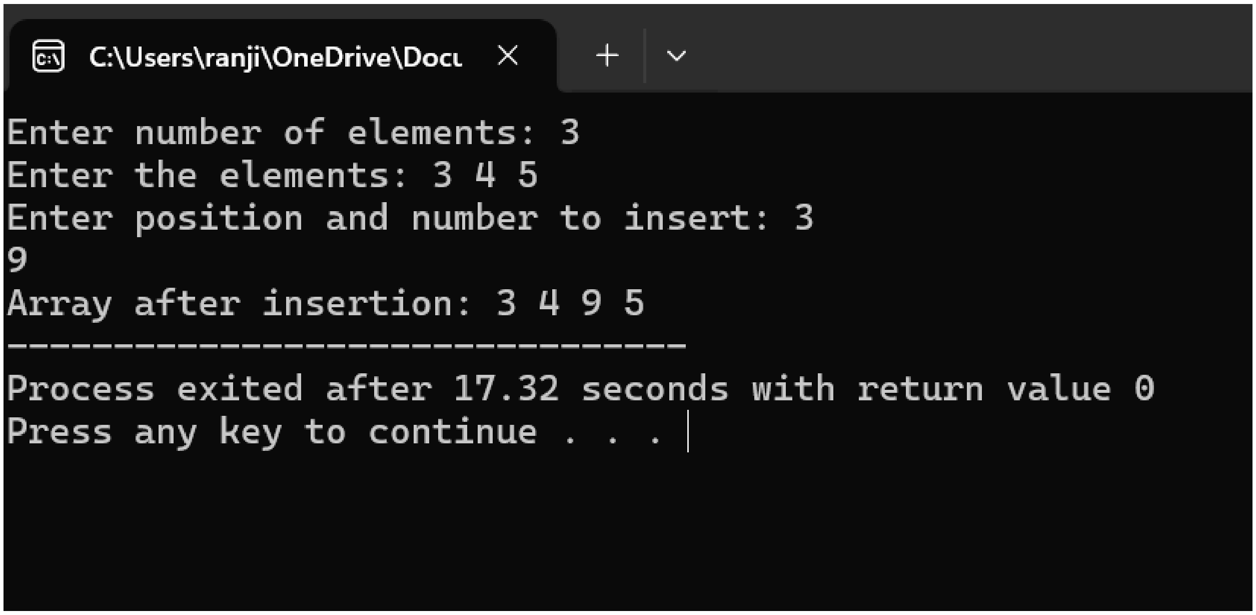


```
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0

--------------------------------
Process exited after 1.762 seconds with return value 0
Press any key to continue . . .
```

DAY-5 (21/09/24)

33. Insert a number in a list:

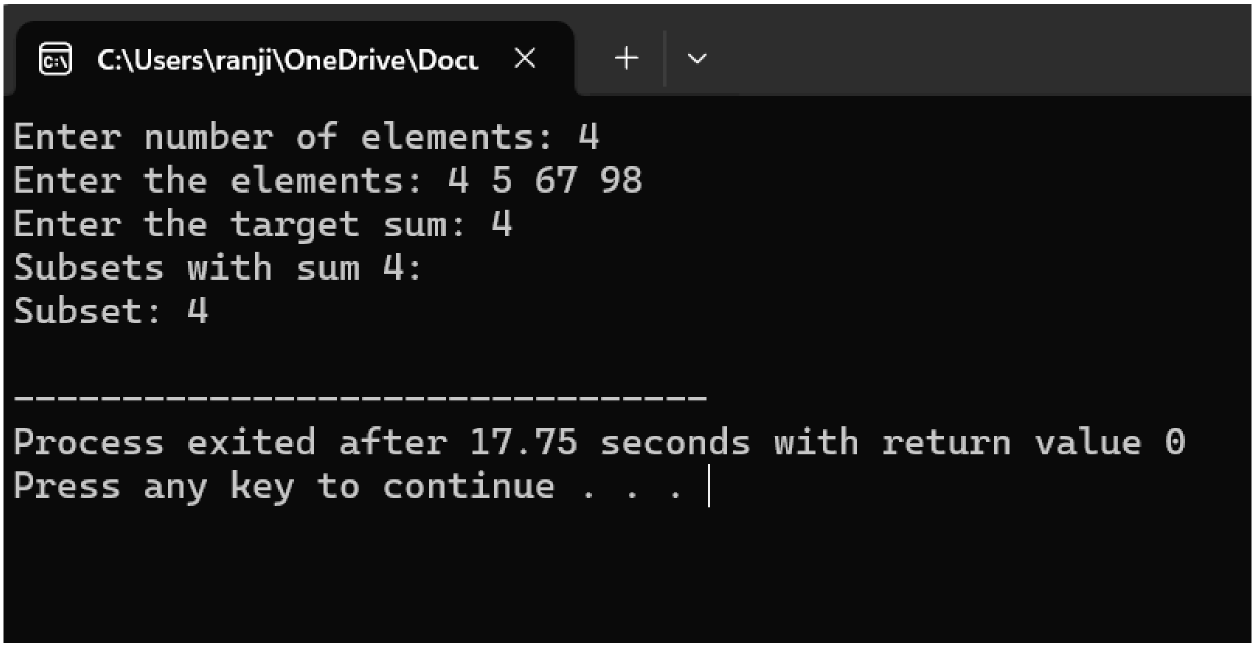
```
#include <stdio.h>
int main() {
    int arr[100], n, pos, num, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter the elements: ");
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    printf("Enter position and number to insert: ");
    scanf("%d %d", &pos, &num);
    for(i = n; i >= pos; i--)
        arr[i] = arr[i - 1];
    arr[pos - 1] = num;
    n++;
    printf("Array after insertion: ");
    for(i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```



```
Enter number of elements: 3
Enter the elements: 3 4 5
Enter position and number to insert: 3
9
Array after insertion: 3 4 9 5
--------------------------------
Process exited after 17.32 seconds with return value 0
Press any key to continue . . .
```

34. Sum of subsets problem using backtracking:

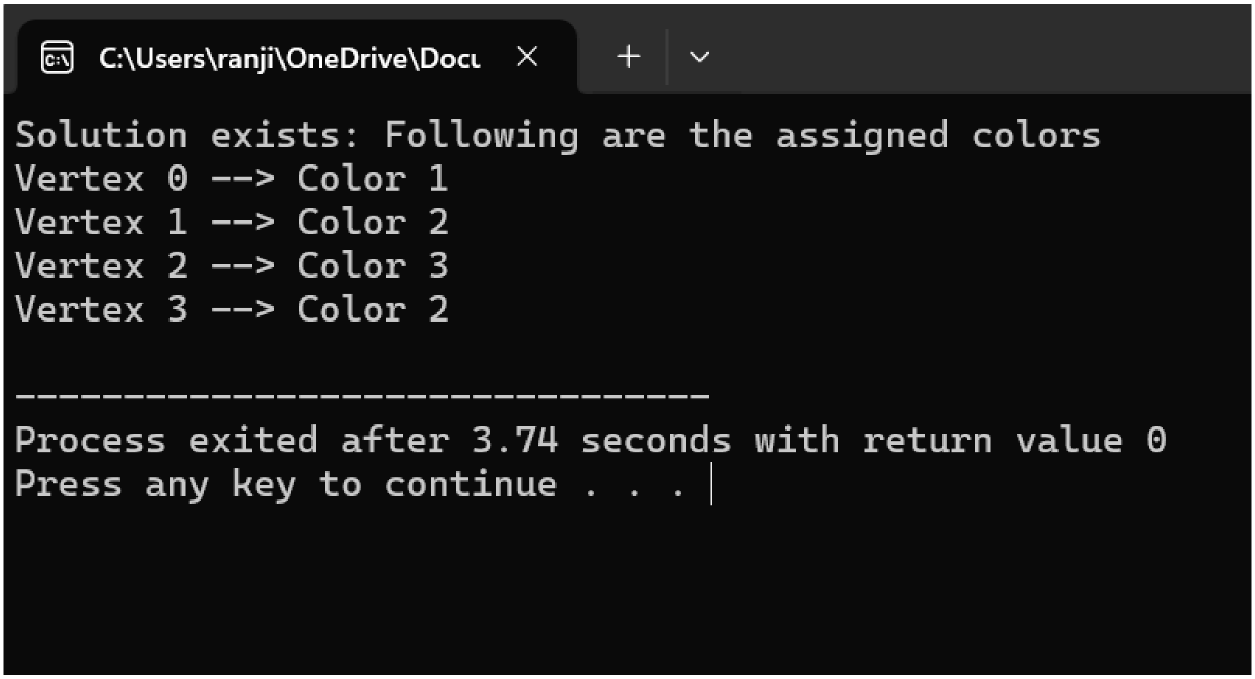
```
#include <stdio.h>
int n, target, subset[100], solution[100];
void sumOfSubsets(int s, int k, int r) {
    solution[k] = 1;
    if (s + subset[k] == target) {
        printf("Subset: ");
        for (int i = 0; i <= k; i++)
            if (solution[i])
                printf("%d ", subset[i]);
        printf("\n");
    } else if (s + subset[k] + subset[k+1] <= target)
        sumOfSubsets(s + subset[k], k + 1, r - subset[k]);
    if (s + r - subset[k] >= target && s + subset[k+1] <= target) {
        solution[k] = 0;
        sumOfSubsets(s, k + 1, r - subset[k]);
    }
}
int main() {
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int total = 0;
    printf("Enter the elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &subset[i]);
        total += subset[i];
    }
    printf("Enter the target sum: ");
    scanf("%d", &target);
    printf("Subsets with sum %d:\n", target);
    sumOfSubsets(0, 0, total);
    return 0;
}
```



```
Enter number of elements: 4
Enter the elements: 4 5 67 98
Enter the target sum: 4
Subsets with sum 4:
Subset: 4

--------------------------------
Process exited after 17.75 seconds with return value 0
Press any key to continue . . .
```

35. Graph coloring using Backtracking:

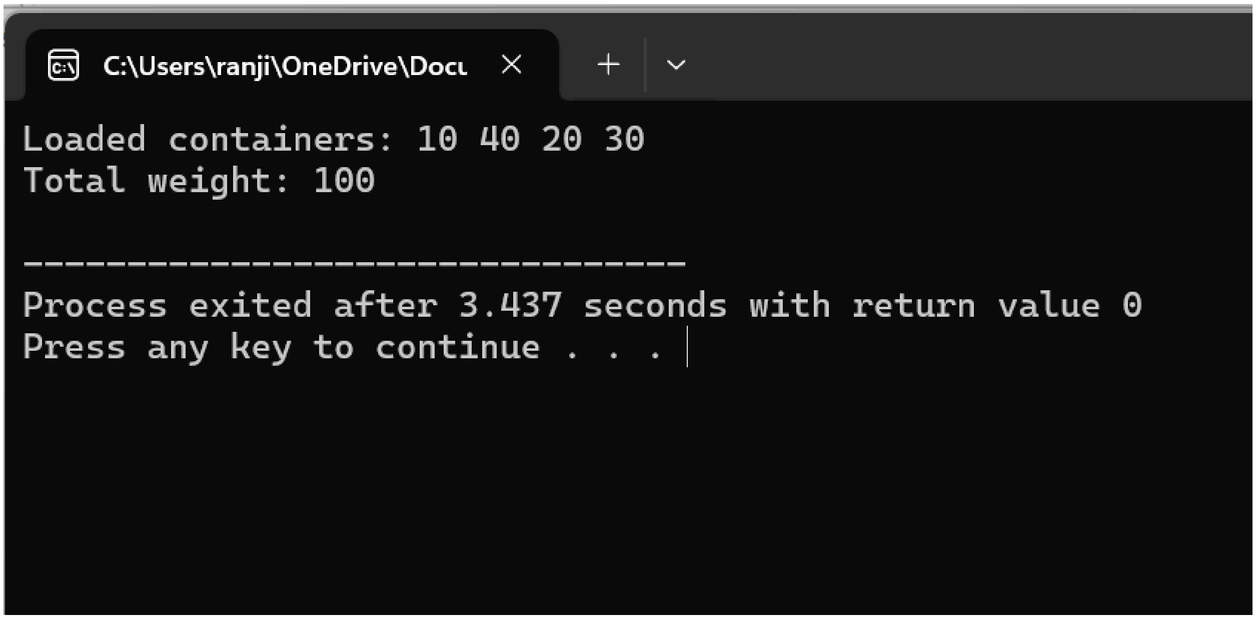
```
#include <stdio.h>
#include <stdbool.h>
#define V 4
bool isSafe(int v, int graph[V][V], int color[], int c) {
    for (int i = 0; i < V; i++) {
        if (graph[v][i] && c == color[i])
            return false;
    }
    return true;
}
bool graphColoringUtil(int graph[V][V], int m, int color[],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

    for (int c = 1; c <= m; c++) {
        if (isSafe(v, graph, color, c)) {
            color[v] = c;

            if (graphColoringUtil(graph, m, color, v + 1))
                return true;

            color[v] = 0;
        }
    }
    return false;
}
bool graphColoring(int graph[V][V], int m) {
    int color[V] = {0};
    if (graphColoringUtil(graph, m, color, 0)) {
        printf("Solution exists: Following are the assigned colors\n");
        for (int i = 0; i < V; i++)
            printf("Vertex %d --> Color %d\n", i, color[i]);
        return true;
    } else {
        printf("Solution does not exist\n");
        return false;
    }
}
int main() {
    int graph[V][V] = {
        {0, 1, 1, 1},
        {1, 0, 1, 0},
        {1, 1, 0, 1},
        {1, 0, 1, 0}
    };
    int m = 3;
    graphColoring(graph, m);
    return 0;
}
```



```
Solution exists: Following are the assigned colors
Vertex 0 --> Color 1
Vertex 1 --> Color 2
Vertex 2 --> Color 3
Vertex 3 --> Color 2

--------------------------------
Process exited after 3.74 seconds with return value 0
Press any key to continue . . .
```

36. Container loader problem:

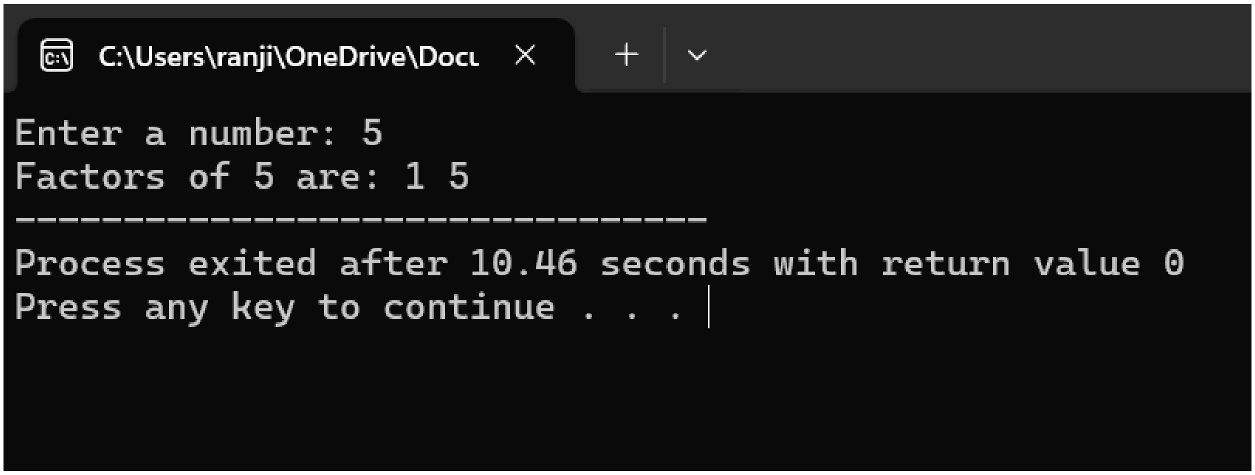
```
#include <stdio.h>
#define MAX_CONTAINERS 100
void loadContainers(int weights[], int n, int maxWeight) {
    int currentWeight = 0;
    printf("Loaded containers: ");
    for (int i = 0; i < n; i++) {
        if (currentWeight + weights[i] <= maxWeight) {
            currentWeight += weights[i];
            printf("%d ", weights[i]);
        }
    }
    printf("\nTotal weight: %d\n", currentWeight);
}
int main() {
    int weights[MAX_CONTAINERS] = {10, 40, 20, 30, 50};
    int n = 5;
    int maxWeight = 100;
    loadContainers(weights, n, maxWeight);
    return 0;
}
```



```
Loaded containers: 10 40 20 30
Total weight: 100

--------------------------------
Process exited after 3.437 seconds with return value 0
Press any key to continue . . .
```

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

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

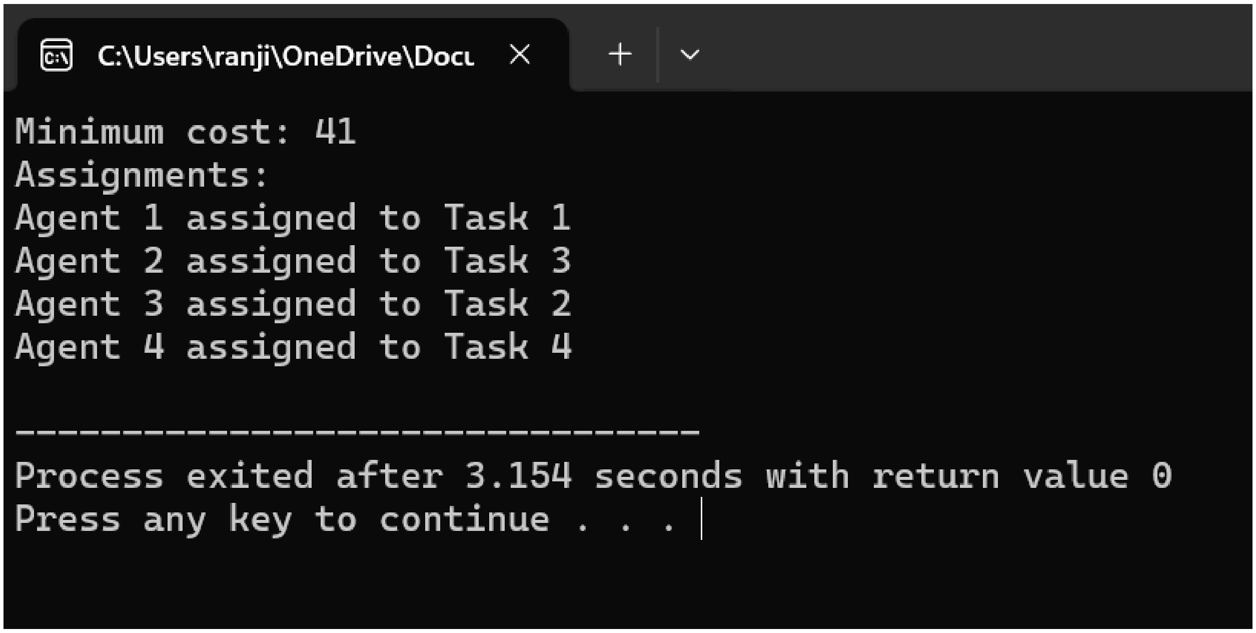


```
Enter a number: 5
Factors of 5 are: 1 5
--------------------------------
Process exited after 10.46 seconds with return value 0
Press any key to continue . . .
```

38. Assignment problem using branch and bound:

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

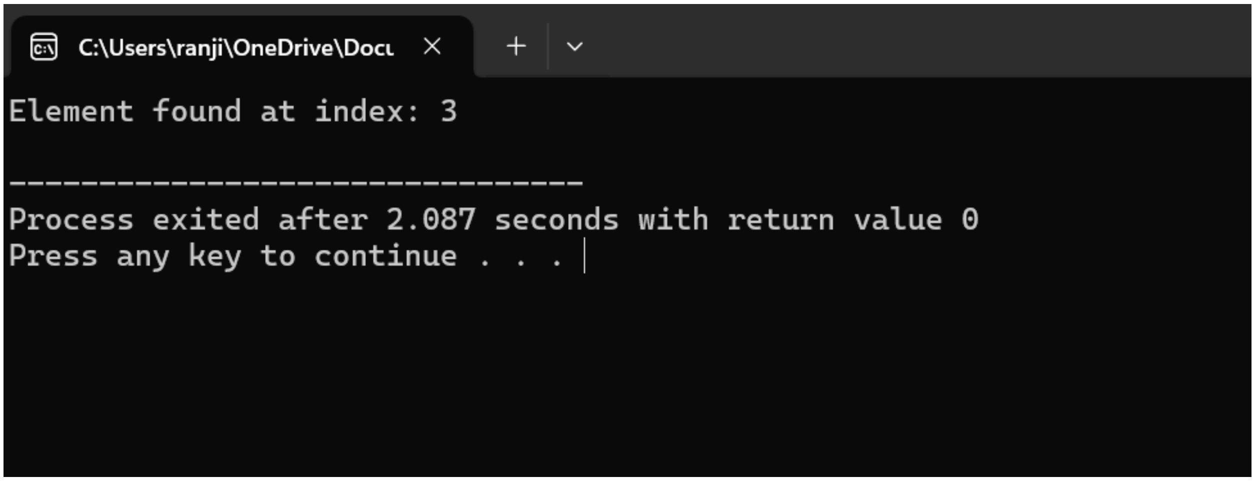
```
#include <limits.h>
#include <stdbool.h>
#define N 4
int cost[N][N] = {
    {10, 19, 8, 15},
    {10, 18, 7, 17},
    {13, 16, 9, 14},
    {12, 19, 20, 8}
};
int final_res = INT_MAX;
int final_assignment[N];
void calculateCost(int assignment[], int n, int current_cost) {
    if (n == N) {
        if (current_cost < final_res) {
            final_res = current_cost;
            for (int i = 0; i < N; i++) {
                final_assignment[i] = assignment[i];
            }
        }
        return;
    }
    for (int i = 0; i < N; i++) {
        bool found = false;
        for (int j = 0; j < n; j++) {
            if (assignment[j] == i) {
                found = true;
                break;
            }
        }
        if (!found) {
            assignment[n] = i;
            calculateCost(assignment, n + 1, current_cost + cost[n][i]);
        }
    }
}
void assignTasks() {
    int assignment[N];
    calculateCost(assignment, 0, 0);
    printf("Minimum cost: %d\n", final_res);
    printf("Assignments:\n");
    for (int i = 0; i < N; i++) {
        printf("Agent %d assigned to Task %d\n", i + 1, final_assignment[i] + 1);
    }
}
int main() {
    assignTasks();
    return 0;
}
```



```
Minimum cost: 41
Assignments:
Agent 1 assigned to Task 1
Agent 2 assigned to Task 3
Agent 3 assigned to Task 2
Agent 4 assigned to Task 4

--------------------------------
Process exited after 3.154 seconds with return value 0
Press any key to continue . . .
```

39. Linear search:

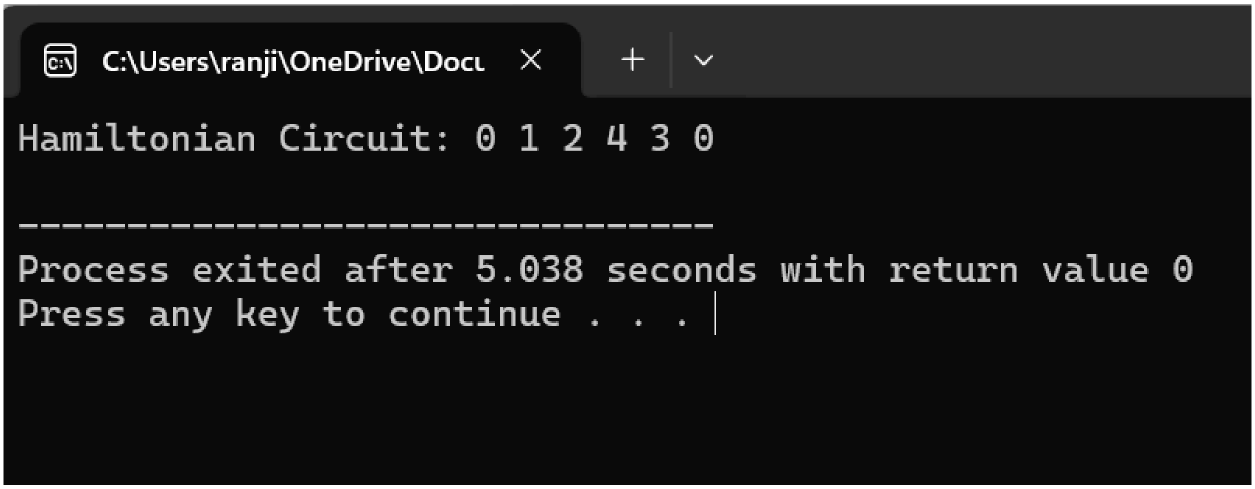
```
#include <stdio.h>
int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i;
        }
    }
    return -1;
}
int main() {
    int arr[] = {5, 3, 8, 4, 2};
    int target = 4;
    int size = sizeof(arr) / sizeof(arr[0]);

    int result = linearSearch(arr, size, target);
    if (result != -1) {
        printf("Element found at index: %d\n", result);
    } else {
        printf("Element not found.\n");
    }

    return 0;
}
```



```
Element found at index: 3

--------------------------------
Process exited after 2.087 seconds with return value 0
Press any key to continue . . .
```

40. Hamiltonian circuit Using backtracking method:

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

            if (hamiltonianUtil(graph, path, pos + 1))
                return true;

            // Remove current vertex if it doesn't lead to a solution
            path[pos] = -1;
        }
    }
    return false;
}
void hamiltonianCircuit(int graph[V][V]) {
    int path[V];
    for (int i = 0; i < V; i++)
        path[i] = -1;
    path[0] = 0;
    if (!hamiltonianUtil(graph, path, 1)) {
        printf("Solution does not exist\n");
        return;
    }
    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, 0, 1, 0},
        {1, 0, 1, 0, 1},
        {0, 1, 0, 1, 1},
        {1, 0, 1, 0, 1},
        {0, 1, 1, 1, 0}
    };
    hamiltonianCircuit(graph);
    return 0;
}
```



```
Hamiltonian Circuit: 0 1 2 4 3 0

--------------------------------
Process exited after 5.038 seconds with return value 0
Press any key to continue . . .
```