CSA0630

DESIGN ANALYSIS AND ALGORITHMS FOR SORTING

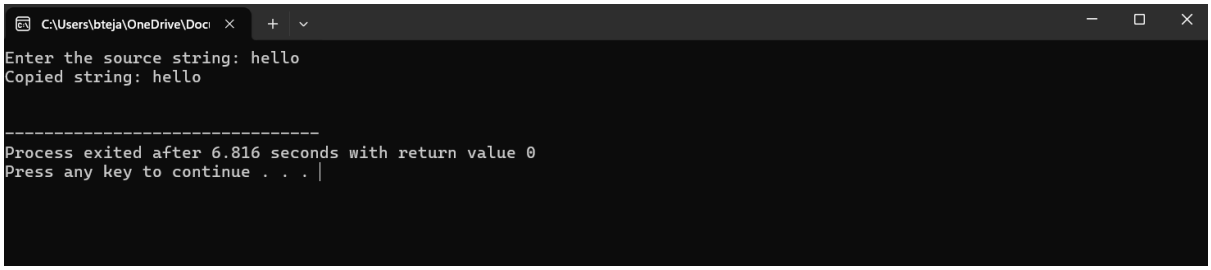
PRACTICAL SESSION DAY2

1. Write a program for to copy one string to another using recursion

PROGRAM:

```
#include <stdio.h>
void copyString(char source[], char destination[], int index) {
    if (source[index] == '\0') {
        destination[index] = '\0';
        return;
    }
    destination[index] = source[index];
    copyString(source, destination, index + 1);
}
int main() {
    char source[100], destination[100];
    printf("Enter the source string: ");
    fgets(source, sizeof(source), stdin);
    copyString(source, destination, 0);
    printf("Copied string: %s\n", destination);
    return 0;
}
```

OUTPUT:



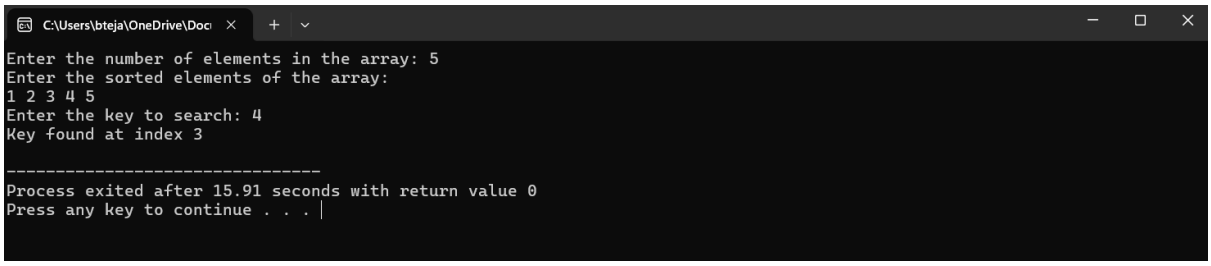
2. Write a Program to perform binary search

PROGRAM:

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

```
    if (result == -1)
        printf("Key not found in the array\n");
    else
        printf("Key found at index %d\n", result);
    return 0;
}
```

OUTPUT:



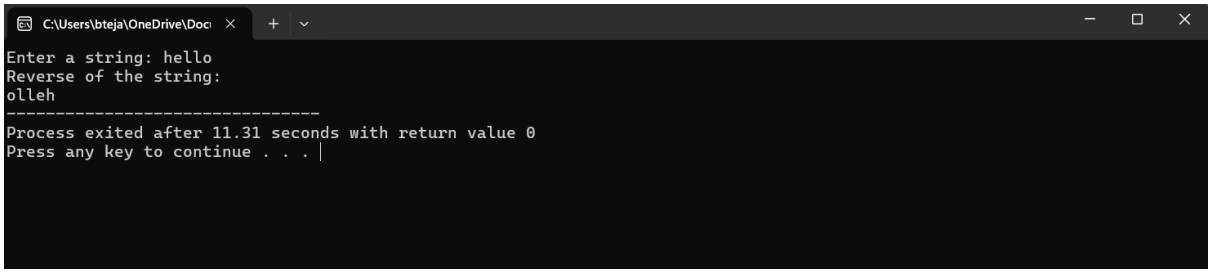
3. Write a program to print the reverse of a string using recursion

PROGRAM:

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

```
    printf("Reverse of the string: ");
    printReverse(str, length);
    return 0;
}
```

OUTPUT:



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

PROGRAM:

```
#include<stdio.h>
int main()
{
        int n,a[100],i,min,max;
        printf("enter the size of array:");
        scanf("%d",&n);
        printf("\nenter the elements in array:");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        min=a[0];
        max=a[0];
        for(i=0;i<n;i++)
        {
                if(a[i]<min)
                {
```

```
                min=a[i];
            }
            if(a[i]>max)
            {
                max=a[i];
            }
        }
        printf("\nminimum element is %d",min);
        printf("\nmaximum element is %d",max);
        return 0;
}
```

OUTPUT:

5. Write a program to perform Strassen’s Matrix Multiplication

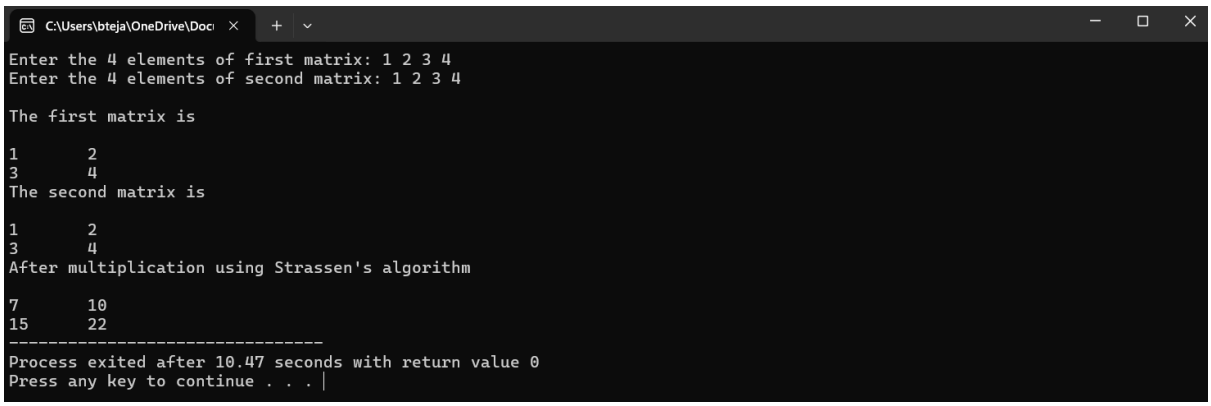
PROGRAM:

```
#include<stdio.h>
int main(){
  int a[2][2], b[2][2], c[2][2], i, j;
  int m1, m2, m3, m4 , m5, m6, m7;
  printf("Enter the 4 elements of first matrix: ");
  for(i = 0;i < 2; i++)
     for(j = 0;j < 2; j++)
         scanf("%d", &a[i][j]);
  printf("Enter the 4 elements of second matrix: ");
  for(i = 0; i < 2; i++)
```

```
    for(j = 0;j < 2; j++)
        scanf("%d", &b[i][j]);
printf("\nThe first matrix is\n");
for(i = 0; i < 2; i++){
    printf("\n");
    for(j = 0; j < 2; j++)
        printf("%d\t", a[i][j]);
}
printf("\nThe second matrix is\n");
for(i = 0;i < 2; i++){
    printf("\n");
    for(j = 0;j < 2; j++)
        printf("%d\t", b[i][j]);
}
m1= (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
m2= (a[1][0] + a[1][1]) * b[0][0];
m3= a[0][0] * (b[0][1] - b[1][1]);
m4= a[1][1] * (b[1][0] - b[0][0]);
m5= (a[0][0] + a[0][1]) * b[1][1];
m6= (a[1][0] - a[0][0]) * (b[0][0]+b[0][1]);
m7= (a[0][1] - a[1][1]) * (b[1][0]+b[1][1]);
c[0][0] = m1 + m4- m5 + m7;
c[0][1] = m3 + m5;
c[1][0] = m2 + m4;
c[1][1] = m1 - m2 + m3 + m6;
 printf("\nAfter multiplication using Strassen's algorithm \n");
 for(i = 0; i < 2 ; i++){
    printf("\n");
    for(j = 0;j < 2; j++)
        printf("%d\t", c[i][j]);
```

```
  }
  return 0;
}
```

OUTPUT:



6. Write a program to perform Merge Sort

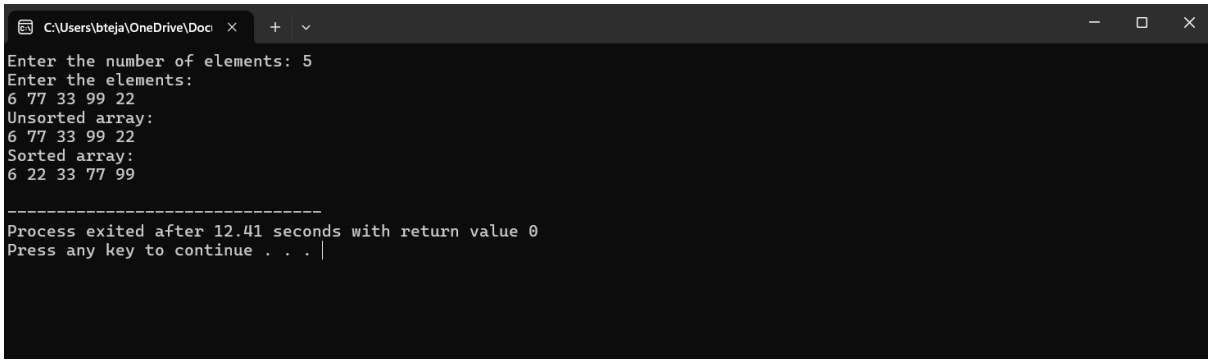
PROGRAM:

```
#include <stdio.h>
void merge(int arr[], int left, int mid, int right) {
  int size1 = mid - left + 1;
  int size2 = right - mid;
  int Left[size1], Right[size2];
  for (int i = 0; i < size1; i++)
    Left[i] = arr[left + i];
  for (int j = 0; j < size2; j++)
    Right[j] = arr[mid + 1 + j];
  int i = 0, j = 0, k = left;
  while (i < size1 && j < size2) {
    if (Left[i] <= Right[j]) {
      arr[k] = Left[i];
      i++;
    } else {
      arr[k] = Right[j];
```

```
      j++;
    }
    k++;
  }
  while (i < size1) {
    arr[k] = Left[i];
    i++;
    k++;
  }
  while (j < size2) {
    arr[k] = Right[j];
    j++;
    k++;
  }
}
void mergeSort(int arr[], int left, int right) {
  if (left < right) {
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);
    merge(arr, left, mid, right);
  }
}
void printArray(int arr[], int n) {
  for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
  printf("\n");
}
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]);
  printf("Unsorted array: \n");
  printArray(arr, n);
  mergeSort(arr, 0, n - 1);
  printf("Sorted array: \n");
  printArray(arr, n);
  return 0;
}
```

OUTPUT:



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

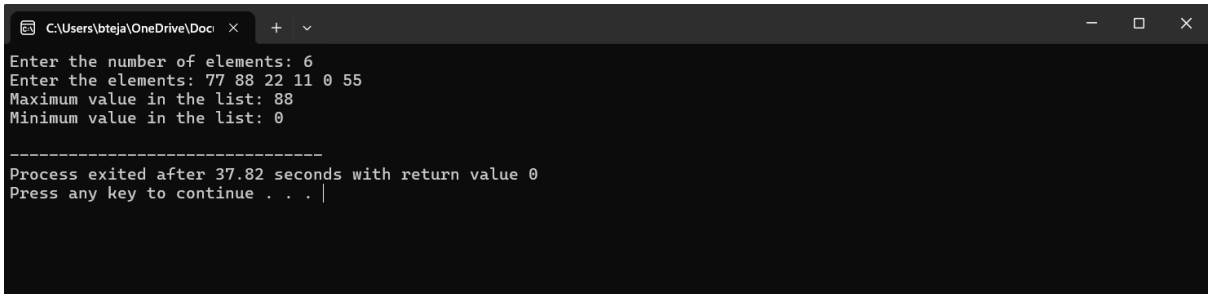
PROGRAM:

```
#include <stdio.h>
struct MaxMin {
    int max;
    int min;
};
struct MaxMin findMaxMin(int arr[], int low, int high) {
    struct MaxMin result, left, right, mid;
```

```
    int midIndex;
    if (low == high) {
        result.max = arr[low];
        result.min = arr[low];
        return result;
    }
    if (high == low + 1) {
        if (arr[low] > arr[high]) {
            result.max = arr[low];
            result.min = arr[high];
        } else {
            result.max = arr[high];
            result.min = arr[low];
        }
        return result;
    }
    midIndex = (low + high) / 2;
    left = findMaxMin(arr, low, midIndex);
    right = findMaxMin(arr, midIndex + 1, high);
    if (left.max > right.max)
        result.max = left.max;
    else
        result.max = right.max;
    if (left.min < right.min)
        result.min = left.min;
    else
        result.min = right.min;
    return result;
}
int main() {
```

```
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    struct MaxMin result = findMaxMin(arr, 0, n - 1);
    printf("Maximum value in the list: %d\n", result.max);
    printf("Minimum value in the list: %d\n", result.min);
    return 0;
}
```

OUTPUT:



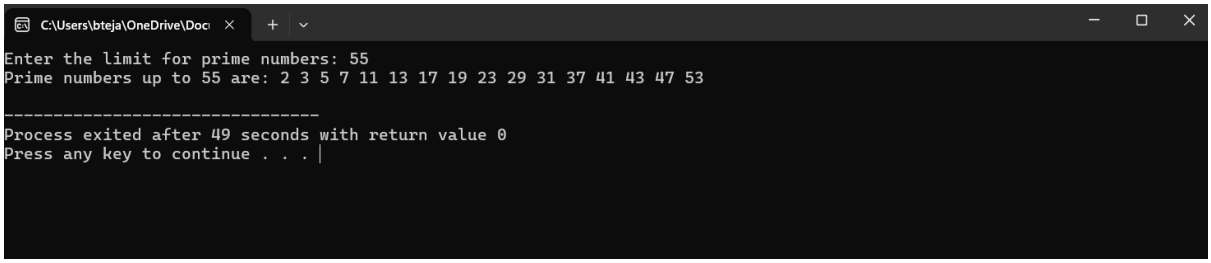
8. Write a program to generate all the prime numbers using recursion

PROGRAM:

```
#include <stdio.h>
#include <stdbool.h>
bool isPrime(int num, int divisor) {
    if (num <= 1) {
        return false;
    }
    if (divisor == 1) {
        return true;
    }
    if (num % divisor == 0) {
        return false;
    }
    return isPrime(num, divisor - 1);
```

```
}
void generatePrimes(int limit, int current) {
   if (current <= limit) {
      if (isPrime(current, current - 1)) {
         printf("%d ", current);
      }
      generatePrimes(limit, current + 1);
   }
}
int main() {
   int limit;
   printf("Enter the limit for prime numbers: ");
   scanf("%d", &limit);
   printf("Prime numbers up to %d are: ", limit);
   generatePrimes(limit, 2);
   printf("\n");
   return 0;
}
```

OUTPUT:



9. Write a program to perform Knapsack problem using greedy techniques

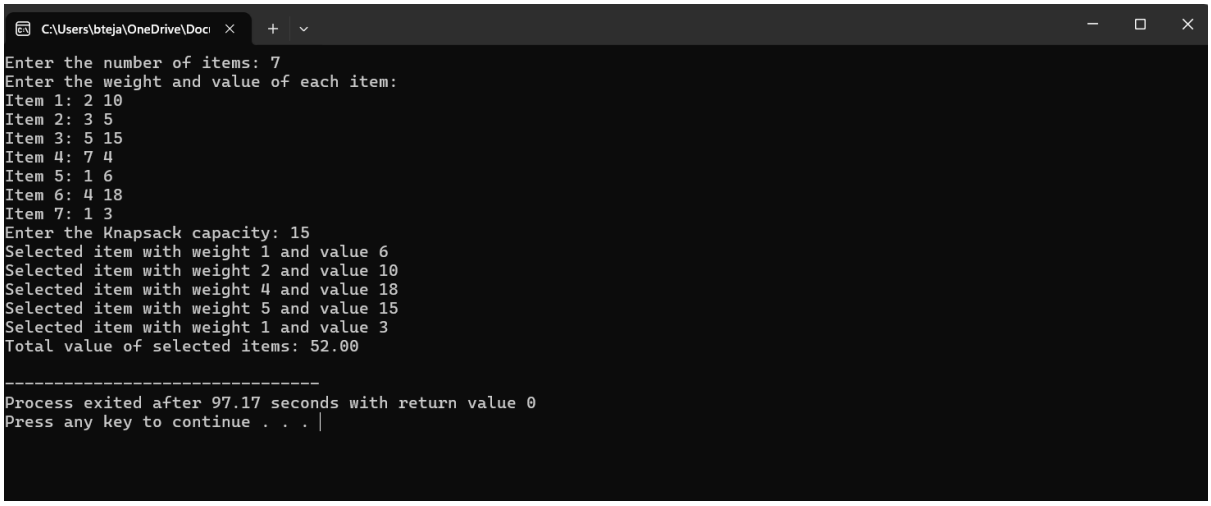
PROGRAM:

```
#include <stdio.h>

void knapsackGreedy(int weights[], int values[], double ratios[], int n, int capacity) {

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

      ratios[i] = (double)values[i] / weights[i];

   }

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

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

         if (ratios[j] < ratios[j + 1]) {

            int tempWeight = weights[j];

            weights[j] = weights[j + 1];

            weights[j + 1] = tempWeight;
```

```
            int tempValue = values[j];
            values[j] = values[j + 1];
            values[j + 1] = tempValue;

            double tempRatio = ratios[j];
            ratios[j] = ratios[j + 1];
            ratios[j + 1] = tempRatio;
        }
    }
}
int currentWeight = 0;
double totalValue = 0.0;
for (int i = 0; i < n; i++) {
    if (currentWeight + weights[i] <= capacity) {
        currentWeight += weights[i];
        totalValue += values[i];
        printf("Selected item with weight %d and value %d\n", weights[i], values[i]);
    }
}
printf("Total value of selected items: %.2f\n", totalValue);
}
int main() {
int n, capacity;
printf("Enter the number of items: ");
scanf("%d", &n);
int weights[n];
int values[n];
double ratios[n];
printf("Enter the weight and value of each item:\n");
```

```
    for (int i = 0; i < n; i++) {
        printf("Item %d: ", i + 1);
        scanf("%d %d", &weights[i], &values[i]);
    }
    printf("Enter the Knapsack capacity: ");
    scanf("%d", &capacity);
    knapsackGreedy(weights, values, ratios, n, capacity);
    return 0;
}
```

OUTPUT:



10. Write a program to perform MST using greedy techniques

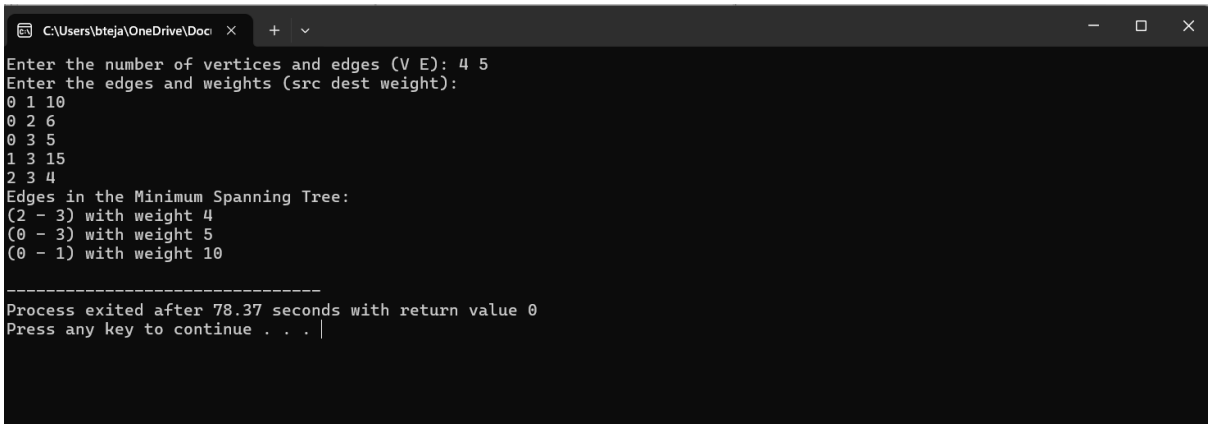
PROGRAM:

```
#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 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++;
    }
}
int compare(const void* a, const void* b) {
    return ((struct Edge*)a)->weight - ((struct Edge*)b)->weight;
}
void kruskalMST(struct Edge edges[], int V, int E) {
    struct Edge result[V];
    int e = 0;
    int i = 0;
    qsort(edges, E, sizeof(edges[0]), compare);
    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;
    }
    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");
    for (i = 0; i < e; i++) {
        printf("(%d - %d) with weight %d\n", result[i].src, result[i].dest, result[i].weight);
    }
    free(subsets);
}
int main() {
    int V, E;
    printf("Enter the number of vertices and edges (V E): ");
    scanf("%d %d", &V, &E);
    struct Edge edges[E];
    printf("Enter the edges and weights (src dest weight):\n");
    for (int i = 0; i < E; i++) {
        scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);
    }
    kruskalMST(edges, V, E);

    return 0;
}
```

OUTPUT:


C:\Users\bteja\OneDrive\Doc
Enter the number of vertices and edges (V E): 4 5
Enter the edges and weights (src dest weight):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges in the Minimum Spanning Tree:
(2 - 3) with weight 4
(0 - 3) with weight 5
(0 - 1) with weight 10

Process exited after 78.37 seconds with return value 0
Press any key to continue . . .