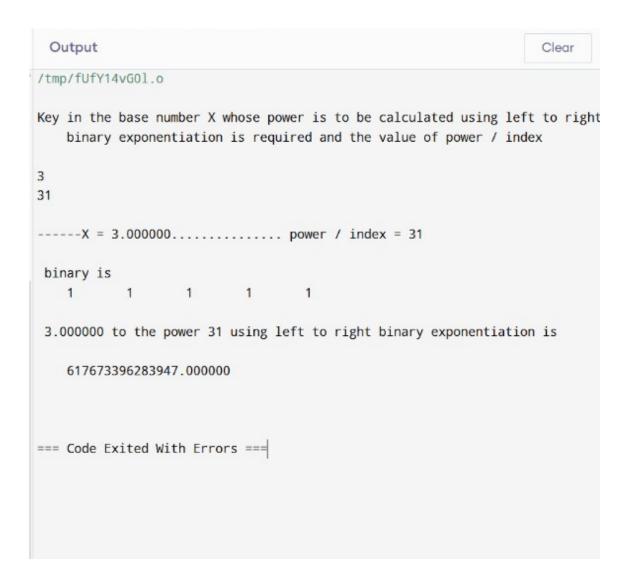
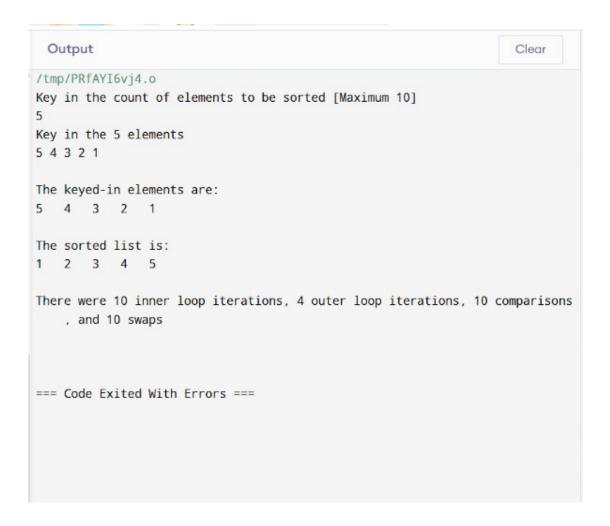
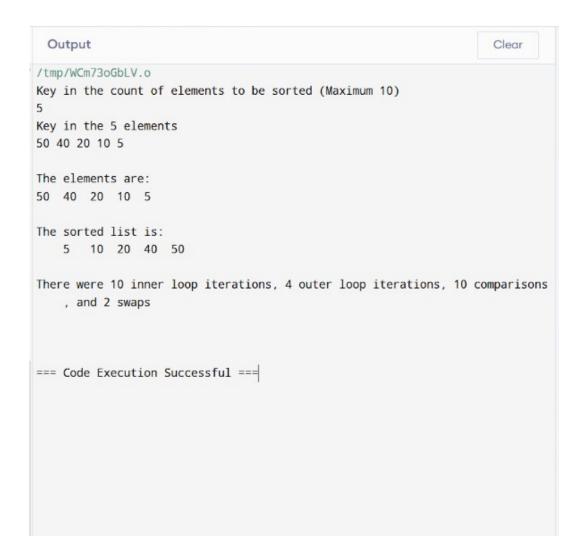


```
Output
                                                                   Clear
/tmp/LGXtaLRET9.o
Key in the degree of polynomial
Key in the 7 coefficients starting from that for degree 6
5
4
-3
2
8
-7
The coefficients are
6 5 4 -3 2 8 -7
Key in the value for X
3
   P(x) at x= 3 is P(3)=5867.
    In the Horner's method:
       [1] The loop was iterated 6 time[s].
       [2] There were 6 multiplication operations.
       [3] There were 6 addition operations.
   In the brute force method:
       [1] P(x) at x=3 is P(3)=5867.
        [2] There were 21 multiplications as against 6 in the Horner's
           method.
=== Code Exited With Errors ===
```

| Output | | Clear |
|---|---|----------|
| /tmp/LNMNaLpxGz.o | | |
| Key in count of items [maximum 50] and the maximum capacity of the bag 5 20 | | |
| <pre>Key in, row-wise [one line per item], the serial number, the weight and the profit for each of the 5 items 1 5 20 2 8 30 3 10 40 4 12 32 5 15 55 The 5 items arranged in non-descending order of the ratio Value = [Profit/Weight] is as under</pre> | | |
| Value Item serial nu | umber Weight Profit | |
| 4.00 1 5 | 5 20 | |
| 4.00 3 1 | 10 40 | |
| 3.75 2 8 | 3 30 | |
| 3.67 5 1 | 15 55 | |
| 2.67 4 1 | 12 32 | |
| The solution to the Fractional Knapsack problem | | |
| Selected Item 1 [whole] | Weight 5 Profit 20 Cumulative Weight 5 Cumulative Valu | e 20.00 |
| Selected Item 3 [whole] | Weight 10 Profit 40 Cumulative Weight 15 Cumulative Valu | e 60.00 |
| Selected Item 2 [part] .75 | Weight 5 Profit 18.75 Cumulative Weight 20 Cumulative | Value 78 |
| Thus the Knapsack with a c | capacity of 20 can hold items worth a Cumulative Total Value of 78. | 75 |
| === Code Exited With Error | -s === | |







```
Output
                                                                                           Clear
/tmp/JMM7wTvXI2.o
Key in the row size and column size [maximum is 10 X 10] for the first matrix, say, A
Key in the row size and column size [maximum is 10 X 10] for the second matrix, say, B
5 2
Key in row-wise the elements of the first [5 \times 5] matrix A
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
First matrix A is:
                3
  1 2
                         4
          2
                 3
          2
   1
                 3
                         4
                                5
          2
   1
                  3
                         4
                                 5
           2
                  3
                         4
Key in row-wise the elements of the second [5 X 2] matrix B
1 10
2 10
3 10
4 10
5 10
Second matrix B is:
   1
          10
          10
   2
   3
         10
         10
   4
          10
The matrix product is C = A \times B and C is:
        150
  55
          150
   55
   55
          150
         150
   55
   55
         150
Statistics:
[1] Outer loop was iterated for 5 time(s).
[2] Middle loop was iterated for 10 time(s).
[3] Inner loop was iterated for 50 time(s).
[4] 50 additions were done.
[5] 50 multiplications were done.
```

Q8) MST Prime algorithm

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
#define TEMP 0
#define PERM 1
#define infinity 9999
#define NIL -1
struct edge
{
  int u;
  int v;
};
int n;
int adj[MAX][MAX];
int predecessor[MAX];
int status[MAX];
int length[MAX];
void create_graph();
void maketree(int r, struct edge tree[MAX]);
int min_temp();
int main()
  int wt_tree = 0;
  int i, root;
  struct edge tree[MAX];
  printf("\n\n vertices are numbered from 0 to end, say, 0, 1,2,...,9; key in edge in the
format <start vertex> <SPACE> <end vertex>, say, without quotes \", \"0 19\". \n\n");
  create_graph();
  printf("\nKey in root vertex: ");
  scanf("%d", &root);
```

```
maketree(root, tree);
  printf("\nEdges to be included in spanning tree are: \n");
for (i = 1; i \le n-1; i++)
  {
    printf("%d->", tree[i].u);
    printf("%d\n", tree[i].v);
    wt_tree += adj[tree[i].u][tree[i].v];
  printf("\nWeight of spanning tree is: %d\n", wt_tree);
  return 0;
/* End of main() */
void maketree(int r, struct edge tree[MAX])
{
  int current, i;
  int count = 0; // Number of vertices in the tree
  /* initialize all vertices */
  for (i = 0; i < n; i++)
    predecessor[i] = NIL;
    length[i] = infinity;
    status[i] = TEMP;
  }
  /* Make length of root vertex 0 */
  length[r] = 0;
  while (1)
  {
    /* search for temporary vertex with minimum length
    and make it current vertex */
    current = min_temp();
    if (current == NIL)
       if (count == n - 1) /* No temporary vertex left */
         return;
       else // Temporary vertices left with length infinity
```

```
{
         printf("\nGraph is not connected, No spanning tree possible");
         exit(1);
       }
    }
    /* Make the current vertex permanent */
    status[current] = PERM;
    /* Insert the edge ( predecessor[current], current) into the tree
    except when the current vertex is root */
    if (current != r)
    {
       count++;
       tree[count].u = predecessor[current];
       tree[count].v = current;
    }
    for (i = 0; i < n; i++)
       if (adj[current][i] > 0 && status[i] == TEMP)
         if (adj[current][i] < length[i])</pre>
            predecessor[i] = current;
           length[i] = adj[current][i];
         }
} /* End of make_tree */
/* Returns the temporary vertex with minimum value of length
    Return NIL if no temporary vertex left or
    all temporary vertices left have path length infinity*/
int min_temp()
{
  int i;
  int min = infinity;
  int k = -1;
  for (i = 0; i < n; i++)
  {
    if (status[i] == TEMP && length[i] < min)
    {
```

```
min = length[i];
       k = i;
     }
  }
  return k;
} /* End of min_temp() */
void create_graph()
  int i, max_edges, origin, destin, wt;
  printf("\nKey in number of vertices: ");
  scanf("%d", &n);
  max_{edges} = n * (n - 1) / 2;
  for (i = 1; i <= max_edges; i++)
  {
     printf("\nKey in edge %d(-1 -1 to quit) : ", i);
     scanf("%d %d", &origin, &destin);
     if ((origin == -1) && (destin == -1))
       break;
     printf("\nKey in weight for that edge: ");
     scanf("%d", &wt);
     if (origin \geq n \mid \mid destin \geq n \mid \mid origin < 0 \mid \mid destin < 0)
     {
       printf("\nInvalid edge\n");
     }
     else
     {
       adj[origin][destin] = wt;
       adj[destin][origin] = wt;
    }
  }
}
```

Compiled using GCC compiler for windows.

```
×
 Windows PowerShell
PS C:\Users\dev\workspace\mca\DAA> gcc .\08_mst_prime_algoritm.c
PS C:\Users\dev\workspace\mca\DAA> .\a.exe
vertices are numbered from 0 to end, say, 0, 1,2,...,9; key in edge in the form at <start vertex> <SPACE> <end vertex>, say, without quotes ", "0 19".
Key in number of vertices: 5
Key in edge 1(-1 -1 \text{ to quit}) : 0 1
Key in weight for that edge: 3
Key in edge 2(-1 -1 \text{ to quit}) : 0 2
Key in weight for that edge: 1
Key in edge 3(-1 -1 \text{ to quit}) : 0 3
Key in weight for that edge: 1
Key in edge 4(-1 -1 to quit) : 0 4
Key in weight for that edge: 2
Key in edge 5(-1 -1 to quit) : 1 2
Key in weight for that edge: 5
Key in edge 6(-1 -1 \text{ to quit}) : 2 3
Key in weight for that edge: 4
Key in edge 7(-1 -1 \text{ to quit}) : 2 4
Key in weight for that edge: 7
Key in edge 8(-1 - 1 \text{ to quit}) : 3 4
Key in weight for that edge: 1
Key in edge 9(-1 -1 to quit) : -1 -1
Key in root vertex: 0
Edges to be included in spanning tree are:
θ−>2
θ−>3
3->4
0->1
Weight of spanning tree is: 6
PS C:\Users\dev\workspace\mca\DAA>
```

Q9) Matrix chain multiplication

// Matrix chain multiplication and how to place the parentheses

```
#include <stdio.h>
#include <limits.h>
#define infinity 99999999
long int m[20][20];
int s[20][20];
int d[20], i, j, n;
void print_optimal(int i, int j)
{
  if (i == j)
     printf("A%d", i);
  else
  {
     printf("(");
     print_optimal(i, s[i][j]);
     print_optimal(s[i][j] + 1, j);
     printf(")");
  }
}
void matmultiply(void)
  long int q;
  int k;
  for (i = n; i > 0; i--)
     for (j = i; j \le n; j++)
     {
       if (i == j)
          m[i][j] = 0;
       else
          for (k = i; k < j; k++)
```

```
q = m[i][k] + m[k + 1][j] + d[i - 1] * d[k] * d[j];
            if (q < m[i][j])
               m[i][j] = q;
              s[i][j] = k;
            }
         }
       }
    }
  }
int MatrixChainOrder(int d[], int i, int j)
{
  if (i == j)
     return 0;
  int k;
  int min = INT_MAX;
  int count;
  for (k = i; k < j; k++)
     count = MatrixChainOrder(d, i, k) +
          MatrixChainOrder(d, k + 1, j) + d[i - 1] * d[k] * d[j];
    if (count < min)
       min = count;
  }
  // Return minimum count
  return min;
void main()
  int k;
  printf("\nKey in the count of matrices\t");
  scanf("%d", &n);
  for (i = 1; i <= n; i++)
    for (j = i + 1; j \le n; j++)
     {
       m[i][j] = 0;
```

```
m[i][j] = infinity;
       s[i][j] = 0;
  printf("\n\nKey in the dimensions\n");
  for (k = 0; k \le n; k++)
     printf("d%d:", k);
    scanf("%d", &d[k]);
  }
  matmultiply();
  printf("\nMinimum cost elements m[i][j] are\n");
  for (i = 1; i \le n; i++)
  {
     printf("\n\n\t");
    for (j = i; j \le n; j++)
       printf("m[%d][%d]:%ld\t", i, j, m[i][j]);
  }
  // cost matrix
  printf("\n\nMinimum cost diagonal matrix is\n");
  for (i = 1; i \le n; i++)
     printf("\n\n");
    for (j = 1; j \le n; j++)
       printf("\t%ld", m[i][j]);
  }
  printf("\n\n Minimum number of multiplications is: %d", MatrixChainOrder(d, 1,
n));
  printf("\n\n");
  i = 1, j = n;
  printf("\n Multiplication Sequence for the %d matrices is ", n);
  print_optimal(i, j);
  printf("\n\n");
```

}

Compiled using GCC compiler for windows.

```
X
                                                                               Windows PowerShell
PS C:\Users\dev\workspace\mca\DAA> gcc .\09_matrix_chain_multiplication.c
PS C:\Users\dev\workspace\mca\DAA> .\a.exe
Key in the count of matrices
                                  5
Key in the dimensions
d0:1
d1:5
d2:4
d3:3
d4:2
d5:1
Minimum cost elements m[i][j] are
        m[1][1]:0
                                                            m[1][4]:38 m[1][5]:40
                          m[1][2]:20
                                           m[1][3]:32
        m[2][2]:0
                          m[2][3]:60
                                           m[2][4]:64
                                                            m[2][5]:38
        m[3][3]:0
                          m[3][4]:24
                                           m[3][5]:18
        m[4][4]:0
                          m[4][5]:6
        m[5][5]:0
Minimum cost diagonal matrix is
        0
                 20
                          32
                                  38
                                           40
        0
                 0
                          60
                                  64
                                           38
        0
                          0
                 0
                                  24
                                           18
                                           6
        0
                 0
                          0
                                  0
        0
                 0
                          0
                                  0
                                           0
 Minimum number of multiplications is : 40
 Multiplication Sequence for the 5 matrices is ((((A1A2)A3)A4)A5)
PS C:\Users\dev\workspace\mca\DAA> |
```