DAA Lab 1

// Chetan Ashok Kachhava
Roll no : 17
Batch : S1

C Program:

```
#include<stdio.h>
void quicksort(int number[25],int first,int last)
 {
        int i, j, pivot, temp;
        if(first<last)
        {
          pivot=first;
          i=first;
          j=last;
            while(i<j)
            {

while(number[i]<=number[pivot]&&i<last)
                i++;
                while(number[j]>number[pivot])
                j--;
                if(i<j)
                {
                   temp=number[i];
                   number[i]=number[j];
                   number[j]=temp;
                }
            }

              temp=number[pivot];
              number[pivot]=number[j];
              number[j]=temp;
              quicksort(number,first,j-1);
              quicksort(number,j+1,last);
          }
       }
```

```
      int main()
      {
          int i, count, number[25];
          printf("How many elements are u going to
enter?: ");
          scanf("%d",&count)
          printf("Enter %d elements: ", count);
          for(i=0;i<count;i++)
              scanf("%d",&number[i]);
                  quicksort(number,0,count-1);
                    printf("Order of Sorted elements:
");
                       for(i=0;i<count;i++)
          printf(" %d",number[i]);
    return 0;
}
```

// Chetan Ashok Kachhava

Roll No : 17

Batch : S1

C program :

```
#include <stdio.h>
int binarySearch(int array[], int x, int low, int high)
   {
            if (high >= low)
             {
                     int mid = (low + high)/2;
                 if (array[mid] == x)
                           return mid;
                if (array[mid] > x)
                          return binarySearch(array, x, low, mid - 1);
                          return binarySearch(array, x, mid + 1, high);
             }
                return -1;
     }
int main(void)
{
```

```
            int array[] = {3, 4, 5, 6, 7, 8, 9};

            int n = sizeof(array) / sizeof(array[0]);
            int x;
            printf("Enter the Elemnt you want to search");

            scanf("%d",&x);

            int result = binarySearch(array, x, 0, n - 1);

           if (result == -1)

                  printf("Not found");

          else

                     printf("Element is found at index %d", result);

}
```

DAA LAB : 03

//chetan Ashok Kachhava

Roll no : 17

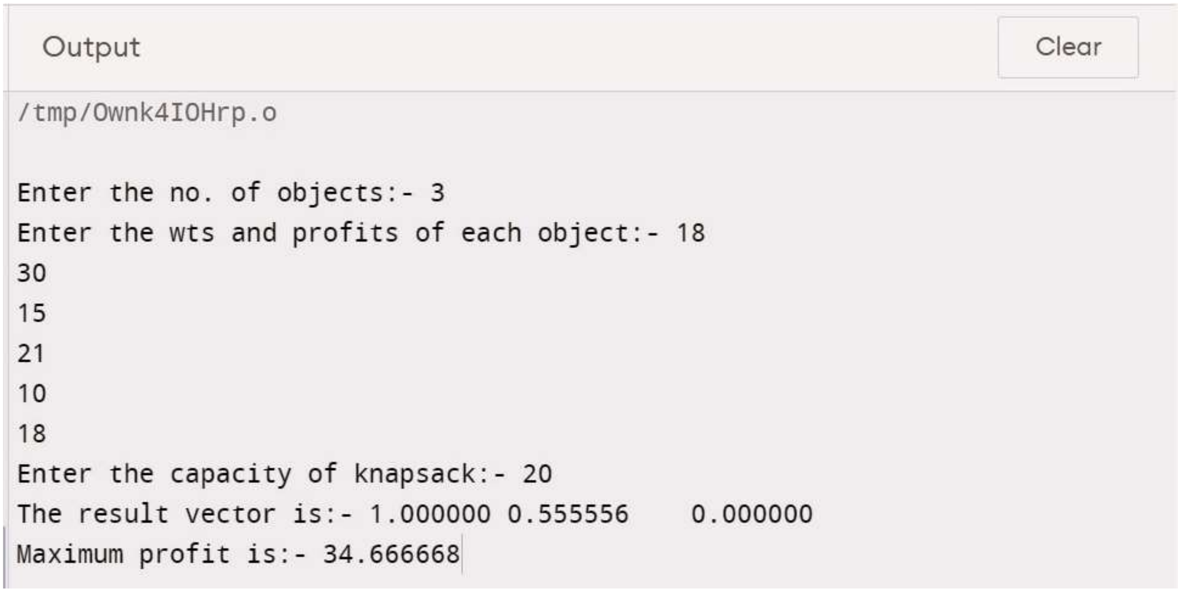
Batch : s1

```
#include<stdio.h>
void
knapsack(int n, float weight[], float profit[], float capacity)

 {

      float x[20], tp = 0;

      int i, j, u;
      u = capacity;
      for (i = 0; i < n; i++)

      x[i] = 0.0;
      for (i = 0; i < n; i++)

      {

          if (weight[i] > u)

         break;

        else

          {

            x[i] = 1.0;

            tp = tp + profit[i];

            u = u - weight[i];
```

```
           }

     }
        if (i < n)

              x[i] = u / weight[i];
              tp = tp + (x[i] * profit[i]);

              printf("\nThe result vector is:- ");

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

                   printf("%f\t", x[i]);

                   printf("\nMaximum profit is:- %f", tp);

}


 int main()

  {

           float weight[20], profit[20], capacity;

           int num, i, j;
           float ratio[20], temp;
           printf("\nEnter the no. of objects:- "); scanf("%d", &
num);

           printf("\nEnter the wts and profits of each object:- ");

          for (i = 0; i < num; i++)

         {

            scanf("%f %f", & weight[i], & profit[i]);

         }

            printf("\nEnter the capacity of knapsack:- ");
```

```
scanf("%f", & capacity);
      for (i = 0; i < num; i++)

         {

           ratio[i] = profit[i] / weight[i];

          }

            for (i = 0; i < num; i++)

             {
                for (j = i + 1; j < num; j++)

                   {

                      if (ratio[i] < ratio[j])

                        {

                            temp = ratio[j];

                            ratio[j] = ratio[i];

                            ratio[i] = temp;


                             temp = weight[j];

                             weight[j] = weight[i];

                             weight[i] = temp;


                             temp = profit[j];

                             profit[j] = profit[i];

                             profit[i] = temp;

                        }
```

```
                    }
                }
        knapsack(num, weight, profit, capacity); return (0);
    }
```



```
/tmp/Ownk4IOHrp.o

Enter the no. of objects:- 3
Enter the wts and profits of each object:- 18
30
15
21
10
18
Enter the capacity of knapsack:- 20
The result vector is:- 1.000000 0.555556    0.000000
Maximum profit is:- 34.666668
```

DAA LAB 04

//chetan ashok kachhava

Roll no : 17

Batch : S1

```
#include<stdio.h>
void knapSack(int W, int n, int val[], int wt[]);
 int getMax(int x, int y);
int main(void)
{
         //the first element is set to -1 as
        //we are storing item from index 1
        //in val[] and wt[] array
        int val[] = {-1, 100, 20, 60, 40}; //value of the items
        int wt[] = {-1, 3, 2, 4, 1}; //weight of the items
        int n = 4; //total items
        int W = 5; //capacity of knapsack
        knapSack(W, n, val, wt);
        return 0;
}
int getMax(int x, int y)
 {
        if(x > y)
            {
                  return x;
             }
             else
            {
               return y;
             }
}

   void knapSack(int Capacity, int n, int val[], int objwt[])
   {
          int i, weight;
          //value table having n+1 rows and W+1 columns
         int V[n+1][Capacity+1];
        //fill the row i=0 with value 0
```

```
    for(weight = 0; weight <= Capacity; weight++)

    {

    V[0][weight] = 0;

    }

    //fille the column w=0 with value 0

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

    {

     V[i][0] = 0;

     }

    //fill the value table

 for(i = 1; i <= n; i++)

      {

          for(weight = 1; weight <= Capacity; weight++)

            {

                if(objwt[i] <= weight)

                {

                 V[i][weight] = getMax(V[i-1][weight], val[i] +
V[i-1][weight - objwt[i]]);

                }

                else

               {

                  V[i][weight] = V[i-1][weight]; }

               }
```

```
                }
        }

//max value that can be put inside the knapsack
printf("Max Value: %d\n", V[n][Capacity]); }
```

/* Name : Chetan ASHOK KACHHAVA
ROLL NO : 17
BATCH : S1

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 four elements of first matrix : ");

   for(i=0;i<2;i++)
      for(j=0;j<2;j++)
         scanf("%d",&a[i][j]);

   printf("Enter the four elements of second matrix : ");
   for(i=0;i<2;i++)
      for(j=0;j<2;j++)
         scanf("%d",&b[i][j]);

   printf("\n The first matrix is \n");
   for(i=0;i<2;i++)
   {
      printf("\n");
      for(j=0;j<2;j++)
         printf("\t%d",a[i][j]);
   }

   printf("\n The second matrix is \n");
   for(i=0;i<2;i++)
   {
      printf("\n");
      for(j=0;j<2;j++)
      printf("\t%d",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+m3-m2+m6;

    printf("\n After Multiplication using strassens algo \n");
    for(i=0;i<2;i++)
    {
        printf("\n");
        for(j=0;j<2;j++)
            printf("%d\t",c[i][j]);
    }
    return 0;
}
```

/* Name : Chetan ASHOK KACHHAVA
ROLL NO : 17
BATCH : S1

Program:-

```
#include<stdio.h>
const int MAX = 100;
void WarshallTransitiveClosure(int graph[MAX][MAX], int numVert);
int main(void)
{
   int i, j, numVert;
   int graph[MAX][MAX];
   printf("Warshall's Transitive Closure\n");
   printf("Enter the number of vertices : ");
   scanf("%d",&numVert);
   printf("Enter the adjacency matrix :-\n");
   for (i=0; i<numVert; i++)
      for (j=0; j<numVert; j++)
         scanf("%d",&graph[i][j]);
   WarshallTransitiveClosure(graph, numVert);
   printf("\nThe transitive closure for the given graph is :-\n");
   for (i=0; i<numVert; i++)
   {
      for (j=0; j<numVert; j++)
      {
         printf("%d\t",graph[i][j]);
      }
      printf("\n");
   }
   return 0;
}

void WarshallTransitiveClosure(int graph[MAX][MAX], int numVert)
{
   int i,j,k;
   for (k=0; k<numVert; k++)
   {
      for (i=0; i<numVert; i++)
      {
         for (j=0; j<numVert; j++)
         {
                if (graph[i][j] || (graph[i][k] && graph[k][j]))
                    graph[i][j] = 1;
            }
        }
    }
}
```

/* Name : Rushikesh Jitendra Badgujar
Roll no : 04
Batch : S1
Floyd-Warshall Algorthim

Program:-

```
#include <stdio.h>
#define nV 4
#define INF 999
void printMatrix(int matrix[][nV]);

void floydWarshall(int graph[][nV]) {
  int matrix[nV][nV], i, j, k;
  for (i = 0; i < nV; i++)
    for (j = 0; j < nV; j++)
      matrix[i][j] = graph[i][j];
  // Adding vertices individually
  for (k = 0; k < nV; k++) {
    for (i = 0; i < nV; i++) {
      for (j = 0; j < nV; j++) {
        if (matrix[i][k] + matrix[k][j] < matrix[i][j])
          matrix[i][j] = matrix[i][k] + matrix[k][j];
      }
    }
  }
  printMatrix(matrix);
}

void printMatrix(int matrix[][nV]) {
  for (int i = 0; i < nV; i++) {
    for (int j = 0; j < nV; j++) {
      if (matrix[i][j] == INF)
        printf("%4s", "INF");
      else
        printf("%4d", matrix[i][j]);
    }
    printf("\n");
  }
}

int main() {
  int graph[nV][nV] = {{0, 3, INF, 5},
            {2, 0, INF, 4},
            {INF, 1, 0, INF},
            {INF, INF, 2, 0}};
  floydWarshall(graph);
}
```

/* Name : ChetanAshok Kachhava
   Roll no : 17
   Batch : S1

Program:-

```
#include <stdio.h>
#define INFINITY 9999
#define MAX 10
void Dijkstra(int Graph[MAX][MAX], int n, int start);

void Dijkstra(int Graph[MAX][MAX], int n, int start) {
   int cost[MAX][MAX], distance[MAX], pred[MAX];
   int visited[MAX], count, mindistance, nextnode, i, j;
   // Creating cost matrix
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         if (Graph[i][j] == 0)
            cost[i][j] = INFINITY;
         else
            cost[i][j] = Graph[i][j];
   for (i = 0; i < n; i++) {
      distance[i] = cost[start][i];
      pred[i] = start;
      visited[i] = 0;
   }
   distance[start] = 0;
   visited[start] = 1;
   count = 1;
   while (count < n - 1) {
      mindistance = INFINITY;
   for (i = 0; i < n; i++)
      if (distance[i] < mindistance && !visited[i]) {
         mindistance = distance[i];
         nextnode = i;
      }
   visited[nextnode] = 1;
   for (i = 0; i < n; i++)
      if (!visited[i])
         if (mindistance + cost[nextnode][i] < distance[i]) {
            distance[i] = mindistance + cost[nextnode][i];
```

```
                pred[i] = nextnode;
            }
        count++;
    }
    // Printing the distance
    for (i = 0; i < n; i++) {
        if (i != start) {
            printf("\n Distance from source to %d: %d", i, distance[i]);
        }
    }
    printf("\n");
}

int main() {
    int Graph[MAX][MAX], i, j, n, u;
    n = 7;

    Graph[0][0] = 0;
    Graph[0][1] = 5;
    Graph[0][2] = 0;
    Graph[0][3] = 0;
    Graph[0][4] = 0;
    Graph[0][5] = 3;
    Graph[0][6] = 10;

    Graph[1][0] = 5;
    Graph[1][1] = 0;
    Graph[1][2] = 2;
    Graph[1][3] = 0;
    Graph[1][4] = 0;
    Graph[1][5] = 18;
    Graph[1][6] = 0;

    Graph[2][0] = 0;
    Graph[2][1] = 2;
    Graph[2][2] = 0;
    Graph[2][3] = 3;
    Graph[2][4] = 18;
    Graph[2][5] = 0;
    Graph[2][6] = 0;

    Graph[3][0] = 0;
    Graph[3][1] = 0;
    Graph[3][2] = 3;
    Graph[3][3] = 0;
    Graph[3][4] = 18;
    Graph[3][5] = 0;
    Graph[3][6] = 0;

    Graph[4][0] = 0;
    Graph[4][1] = 0;
    Graph[4][2] = 0;
    Graph[4][3] = 18;
    Graph[4][4] = 0;
    Graph[4][5] = 5;
    Graph[4][6] = 0;

    Graph[5][0] = 3;
    Graph[5][1] = 18;
    Graph[5][2] = 0;
    Graph[5][3] = 0;
    Graph[5][4] = 5;
    Graph[5][5] = 0;
    Graph[5][6] = 7;

    Graph[6][0] = 10;
    Graph[6][1] = 0;
    Graph[6][2] = 0;
    Graph[6][3] = 0;
    Graph[6][4] = 0;
    Graph[6][5] = 7;
    Graph[6][6] = 0;

    u = 0;
    Dijkstra(Graph, n, u);
    return 0;
}
```

```
/* Name : ChetanAshok Kachhava
   Roll no : 17
   Batch : S1
```

Program :

```
 #include<stdio.h>

  int main()

{

    int cost[10][10],visited[10]={0},i,j,n,no_e=1,min,a,b,min_cost=0;

    printf("Enter number of nodes ");
    scanf("%d",&n);
    printf("Enter cost in form of adjacency matrix\n");

    //input graph

       for(i=1;i<=n;i++)

         {

             for(j=1;j<=n;j++)

                 {

                      scanf("%d",&cost[i][j]);
                     // cost is 0 then initialize it by maximum value
if(cost[i][j]==0)

                     cost[i][j]=1000;

                 }

        }
           // logic for finding minimum cost spanning tree

             visited[1]=1; // visited first node

             while(no_e<n)
  {
        min=1000;
      // in each cycle find minimum cost
     for (i = 1;i <=n;i++)
        {
             for (j = 1;j <=n;j++)
               {
                     if (cost[i][j]<min)
                           {
                                if(visited[I]!=0)
                                     {
                                          min = cost[i][j];
                                          a=i;
                                          b=j;
                                     }
                         }
            }
}
// if node is not visited
   if (visited[b]==0)
   {
      printf(“\n%d to %d cost=%d”,a,b,min);
      min_cost=min_cost+min;
```

```
                        no_e++

                    }

                     visited[b]=1;

                     //initalize with maximum value you can also use any other value

                       cost[a][b]=cost[b][a]=1000;

            }

printf("\nminimum weight is %d",min_cost);

   return 0;

}
```