// QuickSort

```
#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,last,j+1);
    }
}
      int main()
    {
         int i,count,number[25];
         printf("How many elements are you 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 are :");

         for(i=0;i<count;i++)
         printf("%d",number[i]);
         return 0;
   }
```

//Binary Search

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

 // If found at mid, then return it
 if (array[mid] == x)
 return mid;
 // Search the left half
 if (array[mid] > x)
 return binarySearch(array, x, low, mid - 1);
 // Search the right half
 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);
}
```

//Fractional Knapsack

```
# 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);
}
```

//0/1 knapsack problem

```
#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]);
}
```

//Strassens Matrix multipilaction

```
#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;
}
```

//Warshall Algorithm

```
# include <stdio.h>
# include <conio.h>
int n,a[10][10],p[10][10];
void path()
{
    int i,j,k;
    for(i=0;i<n;i++)
       for(j=0;j<n;j++)
           p[i][j]=a[i][j];

    for(k=0;k<n;k++)
         for(i=0;i<n;i++)
               for(j=0;j<n;j++)
                    if(p[i][k]==1 && p[k][j]==1)
                     p[i][j]=1;
}

void main()
{
int i,j;
printf("Enter the number of nodes:");
scanf("%d",&n);
printf("\snEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
      for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
            path();
printf("\nThe path matrix is shown belown\n");
for(i=0;i<n;i++)
 {
      for(j=0;j<n;j++)
      printf("%d ",p[i][j]);
      printf("\n");
  }

}
```

//Floyd warshall Algorithm

```
#include<stdio.h>
#define V 4
#define INF 99999
void printSolution(int dist[][V]);
void floydWarshell(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("Following matrix shows the shortest distances"  " between every pair of
vertices \n");
   int i, j;
   for (i = 0; i < V; i++) {
      for (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 },
                       { INF, 0, 3, INF },
                       { INF, INF, 0, 1 },
                       { INF, INF, INF, 0 }
                     };
   floydWarshell(graph);
   return 0;
}
```

//Dijkstra's Algorithm

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
  int cost[10][10],distance[10],path[10][10],n,v,p,row,column,min,index=1,i,j;
 //use enters no of nodes
 printf("Enter no of nodes : ");
 scanf("%d",&n);                                    //user enters cost of matrix
 printf("Enter cost matrix : ");
 for(i=1;i<=n;i++)
 {
     for(j=1;j<=n;j++)
    {
      scanf("%d",&cost[i][j]);
    }
 }
 //user enters node to be visited
printf("Enter node to visit : ");
scanf("%d",&v);                                    //user enters no of paths for particular node
printf("Enter paths for the selected node : ");
scanf("%d",&p);                                    //path matrix
printf("Enter path matrix \n");
 for(i=1;i<=p;i++)
 {
     for(j=1;j<=n;j++)
    {
      scanf("%d",&path[i][j]);
   }
 }
 for(i=1;i<=p;i++)
{
   distance[i]=0;
```

```
   row=1;
   for(j=1;j<n;j++)
{
   if(row!=v)
   {
    //till i visit the last node
     column=path[i][j+1];
     distance[i] = distance[i]+cost[row][column];
   }
    row=column;
   }
}
  //which distance to be considered
   min=distance[1];
   for(i=1;i<=p;i++)
{
   if(distance[i]<=min)
   {
     min=distance[i];
     index=i;
    }
}
printf("min distance is %d\n",min);
printf("min distance path is\n");
for(i=1;i<=n;i++)
{
   if(path[index][i]!=0)
   printf("--->%d", path[index][i]);
} }
```

//Prims Algorithm

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
 int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
        clrscr();
        printf("\nEnter the number of nodes:");
        scanf("%d",&n);
        printf("\nEnter the adjacency matrix:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
                scanf("%d",&cost[i][j]);
                 if(cost[i][j]==0)
                        cost[i][j]=999;
        }
        visited[1]=1;
        printf("\n");
        while(ne < n)
{
                for(i=1,min=999;i<=n;i++)
                for(j=1;j<=n;j++)
                if(cost[i][j]< min)
                if(visited[i]!=0)
{
                        min=cost[i][j];
                        a=u=i;
                        b=v=j;
                }
                if(visited[u]==0 || visited[v]==0)
                {
                        printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                        mincost+=min;
                        visited[b]=1;
                }
                cost[a][b]=cost[b][a]=999;
        }
        printf("\n Minimun cost=%d",mincost);
        getch();
}
```