1. Merge Sort [Iterative]

**Program:**

#include<stdlib.h>

#include<stdio.h>

void merge(int arr[], int l, int m, int r) {

   int i, j, k;

   int n1 = m - l + 1;

   int n2 = r - m;

   int L[n1], R[n2];

   for (i = 0; i < n1; i++)

      L[i] = arr[l + i];

   for (j = 0; j < n2; j++)

      R[j] = arr[m + 1+ j];

   i = 0, j = 0, k = l;

   while (i < n1 && j < n2) {

      if (L[i] <= R[j]) {

         arr[k] = L[i];

         i++;

      } else {

         arr[k] = R[j];

         j++;

      }

      k++;

   }

   while (i < n1) {

      arr[k] = L[i];

      i++;

      k++;

   }

   while (j < n2) {

      arr[k] = R[j];

      j++;

      k++;

   }

}

void iterativeMergeSort(int arr[], int l, int r) {

   if (l < r){

      int mid = l+(r-l)/2;

      iterativeMergeSort(arr, l, mid);

      iterativeMergeSort(arr, mid+1, r);

      merge(arr, l, mid, r);

   }

}

int main(){

   int arr[] = {12, 11, 13, 5, 6, 7};

   int size = sizeof(arr)/sizeof(arr[0]);

   printf("\t\t ITERATIVE MERGE SORT \n");

   printf("Unsorted Array : \t");

   for (int i=0; i < size; i++)

      printf("%d ",arr[i]);

   iterativeMergeSort(arr, 0, size - 1);

   printf("\nSorted array : \t");

   for (int i=0; i < size; i++)

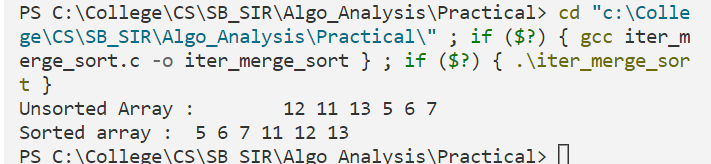
      printf("%d ", arr[i]);

   printf("\n");

   return 0;

}

**Output:**



1. Merge Sort [Recursive]

**Program:**

#include<stdio.h>

void merege(int a[],int low, int mid, int high)

{

    int k,b[50];

    int l=low;

    int i=low;

    int j=mid+1;

    while(l<=mid && j<=high)

    {

        if(a[l]<=a[j])

        {

            b[i]=a[l];

            l=l+1;

        }

        else

        {

            b[i]=a[j];

            j=j+1;

        }

        i++;

    }

    if(l>mid)

    {

        for(int k=j; k<=high;k++)

        {

            b[i]=a[k];

            i++;

        }

    }

    else

    {

        for(int k=l; k<=mid;k++)

        {

            b[i]=a[k];

            i++;

        }

    }

    for(k=low; k<=high;k++)

    {

        a[k]=b[k];

    }

}

void merge\_sort(int a[],int low,int high)

{

    if(low<high)

    {

       int mid=(low+high)/2;

        merge\_sort(a,low,mid);

        merge\_sort(a,mid+1,high);

        merege(a,low,mid,high);

    }

}

int main()

{

    int i, n, a[50];

    printf("Enter size of array: ");

    scanf("%d", &n);

    printf("Eneter elements in array:::");

    for(i=1; i<=n;i++){

        scanf("%d", &a[i]);    }

    merge\_sort(a,1,n);

    printf("After merge sort the arrat is:");

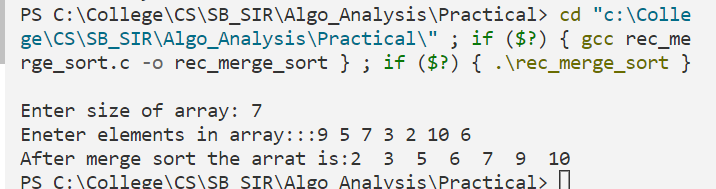
    for(i=1; i<=n; i++){

        printf("%d  ", a[i]);   }

    return 0;

}

**Output:**



1. Quick Sort [Iterative]

**Program:**

#include <stdio.h>

void swap(int\* a, int\* b)

{

    int t = \*a;

    \*a = \*b;

    \*b = t;

}

int partition(int a[], int l, int h)

{

    int x = a[h];

    int i = (l - 1);

    for (int j = l; j <= h - 1; j++) {

        if (a[j] <= x) {

            i++;

            swap(&a[i], &a[j]);

        }

    }

    swap(&a[i + 1], &a[h]);

    return (i + 1);

}

void quick\_sort(int a[], int l, int h)

{

    int stack[h - l + 1];

    int top = -1;

    stack[++top] = l;

    stack[++top] = h;

    while (top >= 0) {

        h = stack[top--];

        l = stack[top--];

        int p = partition(a, l, h);

        if (p  > l) {

            stack[++top] = l;

            stack[++top] = p - 1;

        }

        if (p + 1 < h) {

            stack[++top] = p + 1;

            stack[++top] = h;

        }

    }

}

int main()

{

    int a[50], i, n;

    printf("Enter size of array: ");

    scanf("%d", &n);

    printf("Enter element in arrary:: ");

    for(i=1; i<=n; i++){

        scanf("%d", &a[i]); }

    quick\_sort(a, 1, n);

    printf("After sorting:: ");

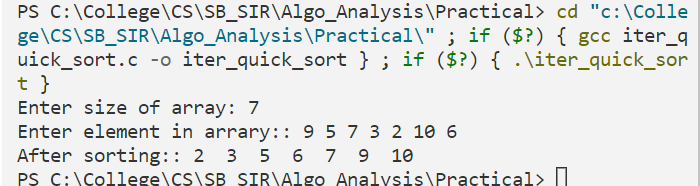
    for(i=1;i<=n; i++){

        printf("%d  ", a[i]);   }

    return 0;

}

**Output:**



1. Quick Sort [Recursive]

**Program:**

#include<stdio.h>

int quick\_sort(int a[], int p, int n)

{

    int pivot=a[p];

    int i=p, j=n+1;

    if(p<n)

    {

    do

    {

        do

        {

            i++;

        }while(a[i]<pivot);

        do

        {

            j--;

        } while (a[j]>pivot);

        if(i<j)

        {

            int temp = a[i];

                a[i] = a[j];

            a[j] = temp;

        }

    }while(i<=j);

    a[p]=a[j];

    a[j]=pivot;

    quick\_sort(a,p,j-1);

    quick\_sort(a,j+1,n);

    }

}

int main()

{

    int i, n, a[50], max=0;

    printf("Enter size of array: ");

    scanf("%d", &n);

    printf("Eneter elements in array:::");

    for(i=1; i<=n;i++)

    {    scanf("%d", &a[i]);    }

    for(i=1; i<=n; i++)

    {

        if(max<a[i])

            max=a[i];

    }

        a[n+1]=max;

   quick\_sort(a,1,n);

    printf("After sort the arrat is:");

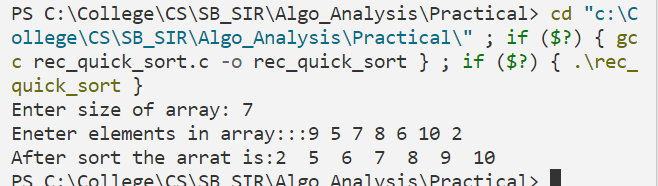
    for(i=1; i<=n;i++)

    {    printf("%d  ", a[i]);  }

    return 0;

}

**Output:**

****

1. Fractional Knapsack

**Program:**

#include<stdio.h>

void knapsack(int n, float capacity, float weight[], float profit[] )

{

    float x[50], tp=0;

    int i,j,u;

    u=capacity;

    for(i=1; i<=n; i++){

        x[i]=0.0;   }

    for(i=1; 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("Result::\n");

    for(i=1; i<=n; i++)

    {    printf("%f\t", x[i]);  }

    printf("Total profit:%f", tp);

}

int main()

{

    float weight[50],profit[50],capacity,ratio[50],temp;

    int num,i,j;

    printf("Enter the no of objects: ");

    scanf("%d", &num);

    printf("Enter the weights and profits of objects:\n");

    for(i=1; i<=num; i++)

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

    printf("Enter the capacity of knapsack: ");

    scanf("%f", &capacity);

    for(i=1; i<=num; i++)

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

    for(i=1; 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;

            }

        }

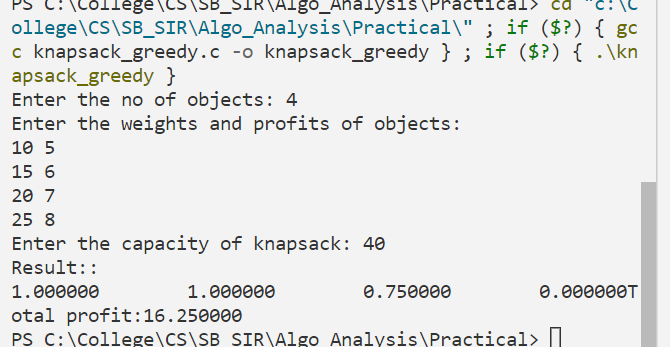
    }

    knapsack(num,capacity,weight,profit);

    return 0;

}

**Output:**



1. 0/1 Knapsack

**Program:**

#include<stdio.h>

int max\_of(int x, int y)

   { return ((x>y)?x:y);    }

void knapsack(int n, int cap, int we[], int p[])

{

    int i,w,v[n+1][cap+1],x;

    for(i=0; i<=n; i++)

    {    v[i][0]=0; }

    for(w=0; w<=cap; w++)

    {    v[0][w]=0; }

    for(i=1; i<=n; i++)

    {

        for(w=0; w<=cap; w++)

        {

            if(we[i-1]<=w)

            {

                if((p[i-1]+v[i-1][w-we[i-1]])>v[i-1][w])

                    v[i][w]=p[i-1]+v[i-1][w-we[i-1]];

                else

                    v[i][w]=v[i-1][w];

            }

            else

                v[i][w]=v[i-1][w];

        }

    }

    for(i=0; i<=n; i++)

    {

        for(w=0; w<=cap; w++)

            printf(" %d ", v[i][w]);

        printf("\n");

    }

    printf("\nProfit-%d", v[n][cap]);

    x=v[n][cap];

    w=cap;

    printf("\nItem taken: ");

    for(i=n; i>0 && x>0; i--)

    {

        if(x==v[i-1][w])

            continue;

        else

        {   printf(" %d ", i);

            x=x-p[i-1];

            w=we[i-1];

        }

    }

}

int main()

{

    int i,n,cap,we[50],p[50];

    printf("Enter no of item: ");

    scanf("%d", &n);

    printf("Enter capacity: ");

    scanf("%d", &cap);

    printf("Enter weight of all items:: ");

    for(i=0; i<n; i++)

    {    scanf("%d", &we[i]);   }

    printf("Enter profit of all items:: ");

    for(i=0; i<n; i++)

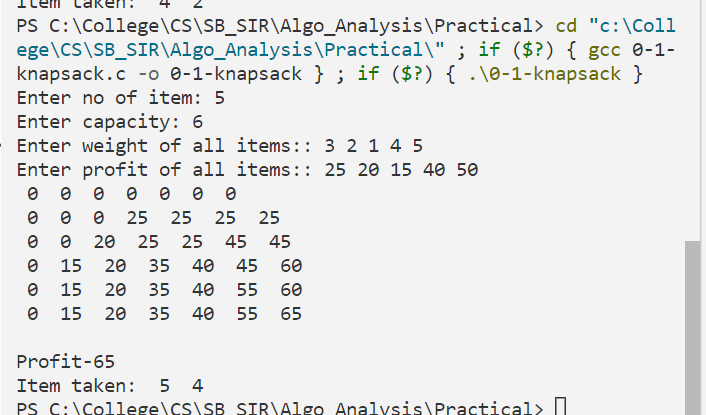
    {    scanf("%d", &p[i]);  }

    knapsack(n,cap,we,p);

    return 0;

}

**Output:**



1. LCS

**Program:**

#include<stdio.h>

#include<string.h>

int max\_of(int u, int v)

{

    int max;

    if(u > v)

        max=u;

    else    max=v;

    return max;

}

int LCS\_length(char x[], char y[])

{

    int m,n,i,j,c[50][50];

    m=strlen(x);

    n=strlen(y);

    for(i=0; i<=m; i++)

        c[i][0]=0;

    for(j=0; j<=n; j++)

        c[0][j]=0;

        for ( i = 1; i <=m; i++)

    {

        for ( j = 1; j <=n; j++)

        {

            if(x[i-1]==y[j-1])

                c[i][j]=c[i-1][j-1] + 1;

            else

                c[i][j]=max\_of(c[i-1][j],c[i][j-1]);

        }

    }

    printf("\n");

    for(i=0; i<=m; i++)

    {

        for(j=0; j<=n; j++)

        {

            printf("%d  ", c[i][j]);

        }

        printf("\n");

    }

    printf("\nMatch string- %d ", c[i-1][j-1]);

    printf("\nLCS is: ");

    int r = c[i-1][j-1];

    char lcs[r+1];

    lcs[r]='\0';

    i=m; j=n;

        while(i>0 && j>0)

        {

            if(x[i-1]==y[j-1])

            {

                lcs[r-1]=x[i-1];

                i--;

                j--;

                r--;

            }

            else if(c[i-1][j] > c[i][j-1])

                i--;

            else

                j--;

        }

    printf("%s", lcs);

   return 0;

}

int main()

{

    char x[100],y[100];

    int i,j,a;

    printf("Enter the 1st string: ");

    scanf("%s", &x);

    printf("Enter the 2nd string: ");

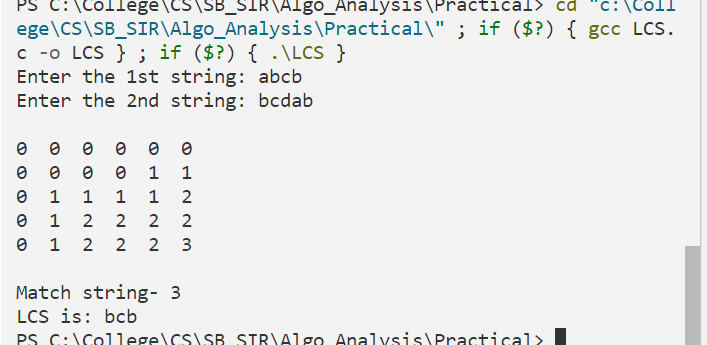
    scanf("%s", &y);

    LCS\_length(x,y);

    return 0;

}

**Output:**



1. Prims Algorithm

**Program:**

#include<stdio.h>

int main()

{

    int i,j,n, weight[50][50];

    int a,b,u,v,min,mincost,visited[50], no\_edge;

    printf("Entef the no of vertices: ");

    scanf("%d", &n);

    printf("Enter weight of each edge::\n");

    for(i=1; i<=n; i++)

    {

        for(j=1; j<=n; j++)

        {

            scanf("%d", &weight[i][j]);

            if (weight[i][j]==0)

                weight[i][j]=999;

        }

    }

    for(i=1; i<=n; i++)

        visited[i]=0;

    printf("the edges considered for MST are");

    visited[1]=1;

    mincost=0;

    while(no\_edge<n)

    {

        for(i=1,min=999; i<=n; i++)

        {

            for(j=1; j<=n; j++)

            {

                if(weight[i][j]<min)

                {

                    if(visited[i]!=0)

                    {

                        min=weight[i][j];

                        a=u=i;

                        b=v=j;

                    }

                }

            }

        }

        if(visited[a]==0 || visited[b]==0)

        {

            printf("\nedge%d(%d %d)  cost=%d", no\_edge++,u,v,min);

            mincost+=min;

            visited[b]=1;

        }

        weight[a][b]=999;

        weight[b][a]=999;

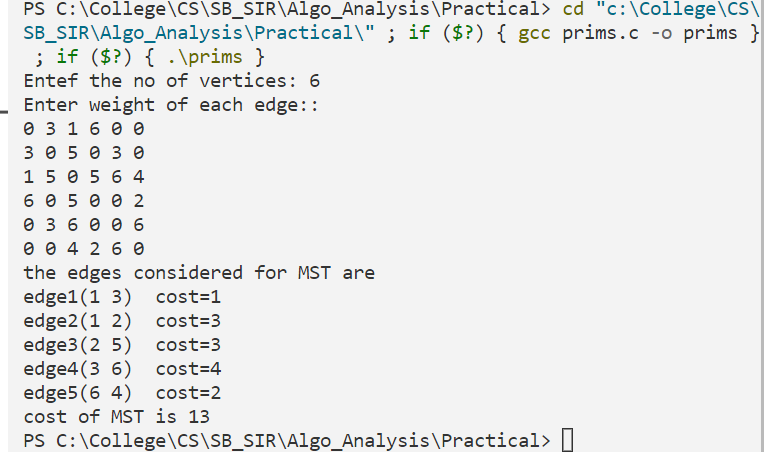
    }

    printf("\ncost of MST is %d", mincost);

    return 0;

}

**Output:**



1. Kruskal

**Program:**

#include<stdio.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,arr[50][50],parent[9];

int find\_parent(int);

void main()

{

    printf("\nEnter the no. of vertices:");

    scanf("%d",&n);

    printf("\nEnter the edge value::\n");

    for(i=1;i<=n;i++)

    {

        for(j=1;j<=n;j++)

        {

            scanf("%d",&arr[i][j]);

            if(arr[i][j]==0)

                arr[i][j]=999;

        }

    }

    printf("The edges of Minimum Cost Spanning Tree are\n");

    while(ne < n)

    {

        for(i=1,min=999;i<=n;i++)

        {

            for(j=1;j <= n;j++)

            {

                if(arr[i][j] < min)

                {

                    min=arr[i][j];

                    a=u=i;

                    b=v=j;

                }

            }

        }

        u=find\_parent(u);

        v=find\_parent(v);

        if(u!=v)

        {

            parent[v]=u;

            printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);

            mincost +=min;

        }

        arr[a][b]=arr[b][a]=999;

    }

    printf("Minimum cost = %d\n",mincost);

}

int find\_parent(int i)

{

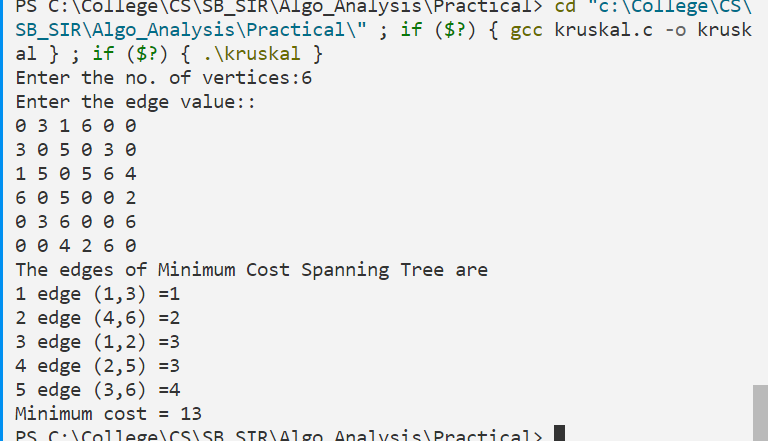
    while(parent[i])

    i=parent[i];

    return i;

}

**Output:**



1. Dijkstra Algorithm

**Program:**

// Dijkstra's Algorithm in C

#include <stdio.h>

#define INFINITY 9999

#define MAX 10

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;

int s=start;

  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++;

  }

  for (i = 0; i < n; i++)

    if (i != start) {

      printf("\nDistance from source %d to %d: %d", s, i, distance[i]);

    }

}

int main() {

  int Graph[MAX][MAX], i, j, n, u;

  n = 7;

  Graph[0][0] = 0;

  Graph[0][1] = 0;

  Graph[0][2] = 1;

  Graph[0][3] = 2;

  Graph[0][4] = 0;

  Graph[0][5] = 0;

  Graph[0][6] = 0;

  Graph[1][0] = 0;

  Graph[1][1] = 0;

  Graph[1][2] = 2;

  Graph[1][3] = 0;

  Graph[1][4] = 0;

  Graph[1][5] = 3;

  Graph[1][6] = 0;

  Graph[2][0] = 1;

  Graph[2][1] = 2;

  Graph[2][2] = 0;

  Graph[2][3] = 1;

  Graph[2][4] = 3;

  Graph[2][5] = 0;

  Graph[2][6] = 0;

  Graph[3][0] = 2;

  Graph[3][1] = 0;

  Graph[3][2] = 1;

  Graph[3][3] = 0;

  Graph[3][4] = 0;

  Graph[3][5] = 0;

  Graph[3][6] = 1;

  Graph[4][0] = 0;

  Graph[4][1] = 0;

  Graph[4][2] = 3;

  Graph[4][3] = 0;

  Graph[4][4] = 0;

  Graph[4][5] = 2;

  Graph[4][6] = 0;

  Graph[5][0] = 0;

  Graph[5][1] = 3;

  Graph[5][2] = 0;

  Graph[5][3] = 0;

  Graph[5][4] = 2;

  Graph[5][5] = 0;

  Graph[5][6] = 1;

  Graph[6][0] = 0;

  Graph[6][1] = 0;

  Graph[6][2] = 0;

  Graph[6][3] = 1;

  Graph[6][4] = 0;

  Graph[6][5] = 1;

  Graph[6][6] = 0;

  printf("Path Matrix is::\n");

  for(i=0; i<n; i++)

  {

    for(j=0; j<n; j++)

    {

        printf(" %d ", Graph[i][j]);

    }printf("\n");

  }

  u = 0;

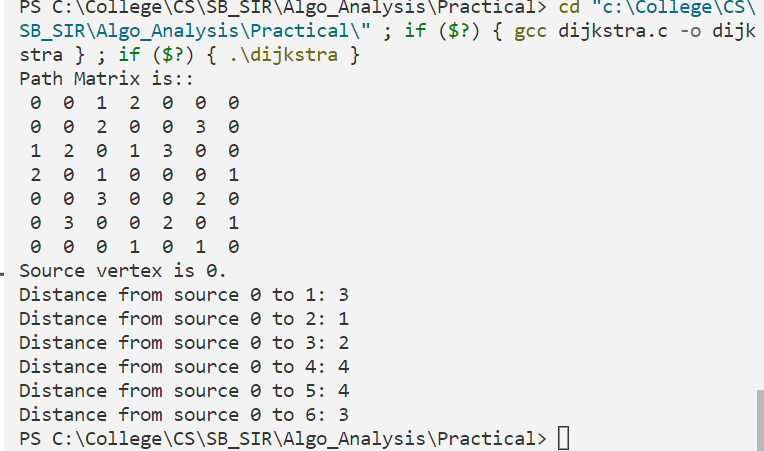
  printf("Source vertex is %d.", u);

  Dijkstra(Graph, n, u);

  return 0;

}

**Output**



1. Binary Search [Iterative]

**Program:**

#include<stdio.h>

int binary\_search(int a[], int l, int h, int x)

{

    while(h>=l)

    {

        int mid=(l+h)/2;

        if (a[mid]==x)

            return mid;

        else if(a[mid]>x)   h=mid-1;

        else  l=mid+1;

    }

    return -1;

}

int main()

{

    int a[50],i,n,x,result;

         printf("Enter the size of array: ");

     scanf("%d", &n);

      printf("Enter value in array in sorted order:::\n");

     for(i=1;i<=n;i++)

     {

        scanf("%d", &a[i]);

     }

      printf("Search value: ");

      scanf("%d", &x);

      result=binary\_search(a,1,n,x);

      if(result==-1)

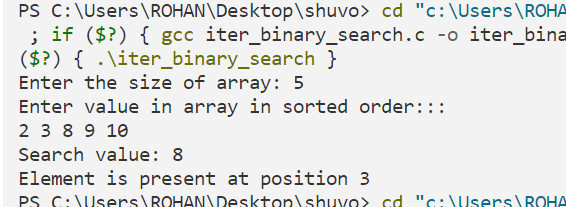
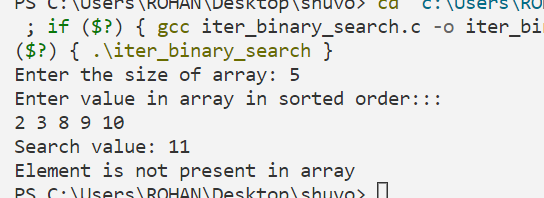
        printf("Element is not present in array");

    else    printf("Element is present at position %d" , result);

    return 0;

}

**Output:**

1. Binary Search [Recursive]

**Program:**

#include<stdio.h>

int binary\_search(int a[], int l, int h, int x)

{

    if(h>=l)

    {

        int mid= (l+h)/2;

        if(a[mid]==x)

            return mid;

        else if(a[mid]>x)

            return binary\_search(a,l,mid-1,x);

        else   return  binary\_search(a,mid+1,h,x);

    }

    return -1;

}

int main()

{

    int a[50],i,n,x,result;

    printf("Enter the size of array: ");

     scanf("%d", &n);

      printf("Enter value in array in sorted order:::");

     for(i=1;i<=n;i++)

     {

        scanf("%d", &a[i]);

     }

      printf("Search value: ");

      scanf("%d", &x);

      result=binary\_search(a,1,n,x);

      if(result==-1)

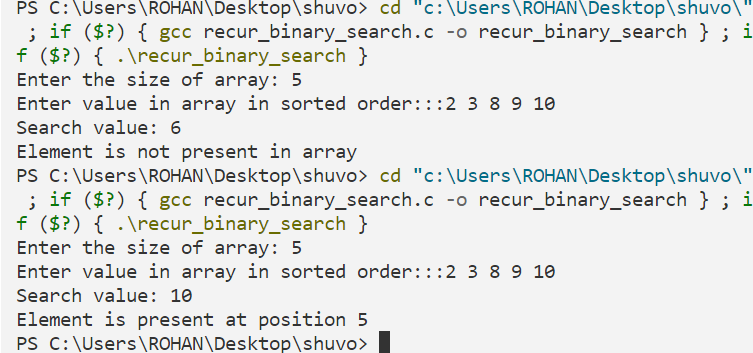
        printf("Element is not present in array");

    else    printf("Element is present at position %d" , result);

    return 0;

}

**Output:**



1. Warshall Algorithm

**Program:**

#include<stdio.h>

int main()

{

    int i,j,k,n,path[50][50],a[50][50];

    printf("Enter no of vertices: ");

    scanf("%d", &n);

    printf("Enter direct path matrix::\n");

    for(i=0; i<n; i++)

    {

        for(j=0; j<n; j++)

        {

            scanf("%d", &path[i][j]);

            if(i==j)

                path[i][j]=0;

        }

    }

    printf("The matrix is::\n");

    for(i=0; i<n; i++)

    {

        for(j=0; j<n; j++)

            printf(" %d ", path[i][j]);

        printf("\n");

    }

    for(i=0; i<n; i++)

    {

       for(j=0; j<n; j++)

            a[i][j]=path[i][j];

    }

    for(k=0; k<n; k++)

    {

       for(i=0; i<n; i++)

        {

            for(j=0; j<n; j++)

            {

                if(a[i][j]!=1)

                    a[i][j]=a[i][j] || a[i][k] && a[k][j];

            }

        }

    }

    printf("The final matrix is::\n");

    for(i=0; i<n; i++)

    {

        for(j=0; j<n; j++)

            printf(" %d ", a[i][j]);

        printf("\n");

    }

    return 0;

}

**Output:**

