```
#include <stdio.h>
#define MAX 50
int p[MAX], w[MAX], x[MAX];
double maxprofit;
int n, m, i;
void greedyKnapsack(int n, int w[], int p[], int m)
{
    double ratio[MAX];

// Calculate the ratio of profit to weight for each item
    for (i = 0; i < n; i++)
    {
        ratio[i] = (double)p[i] / w[i];
    }
// Sort items based on the ratio in non-increasing order
    for (i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (ratio[i] < ratio[j])
            {
                double temp = ratio[i];
                ratio[i] = ratio[j];
                ratio[j] = temp;

                int temp2 = w[i];
                w[i] = w[j];
                w[j] = temp2;

                temp2 = p[i];
                p[i] = p[j];
                p[j] = temp2;
            }
        }
    }
    int currentWeight = 0;
    maxprofit = 0.0;
// Fill the knapsack with items
    for (i = 0; i < n; i++)
    {
        if (currentWeight + w[i] <= m)
        {
            x[i] = 1; // Item i is selected
            currentWeight += w[i];
            maxprofit += p[i];
        }
        else
        {
// Fractional part of item i is selected
            x[i] = (m - currentWeight) / (double)w[i];
            maxprofit += x[i] * p[i];
            break;
        }
    }
    printf("Optimal solution for greedy method: %.1f\n", maxprofit);
    printf("Solution vector for greedy method: ");
    for (i = 0; i < n; i++)
        printf("%d\t", x[i]);
}
int main()
{
    printf("Enter the number of objects: ");
    scanf("%d", &n);
    printf("Enter the objects' weights: ");
    for (i = 0; i < n; i++)
        scanf("%d", &w[i]);
    printf("Enter the objects' profits: ");
    for (i = 0; i < n; i++)
        scanf("%d", &p[i]);
    printf("Enter the maximum capacity: ");
    scanf("%d", &m);
    greedyKnapsack(n, w, p, m);
    return 0;
}
```

2)

```
#include<stdio.h>
#define INF 999
int prim(int c[10][10],int n,int s)
{
    int v[10],i,j,sum=0,ver[10],d[10],min,u;
    for(i=1; i<=n; i++)
    {
        ver[i]=s;
        d[i]=c[s][i];
        v[i]=0;
    }
    v[s]=1;
    for(i=1; i<=n-1; i++)
    {
        min=INF;
        for(j=1; j<=n; j++)
            if(v[j]==0 && d[j]<min)
            {
                min=d[j];
                u=j;
            }
        v[u]=1;
        sum=sum+d[u];
        printf("\n%d -> %d sum=%d",ver[u],u,sum);
        for(j=1; j<=n; j++)
            if(v[j]==0 && c[u][j]<d[j])
            {
                d[j]=c[u][j];
                ver[j]=u;
            }
    }
    return sum;
}
void main()
{
    int c[10][10],i,j,res,s,n;
    printf("\nEnter n value:");
    scanf("%d",&n);
    printf("\nEnter the graph data:\n");
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
            scanf("%d",&c[i][j]);
    printf("\nEnter the souce node:");
    scanf("%d",&s);
    res=prim(c,n,s);
    printf("\nCost=%d",res);
    getch();
}
```

3)

```
#include<stdio.h>
#include<conio.h>
#define INF 999
int min(int a,int b)
{
    return(a<b)?a:b;
}
void floyd(int p[][10],int n)
{
    int i,j,k;
    for(k=1; k<=n; k++)
        for(i=1; i<=n; i++)
            for(j=1; j<=n; j++)
                p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
void main()
{
    int a[10][10],n,i,j;
    printf("\nEnter the n value:");
    scanf("%d",&n);
    printf("\nEnter the graph data:\n");
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
            scanf("%d",&a[i][j]);
    floyd(a,n);
    printf("\nShortest path matrix\n");
    for(i=1; i<=n; i++)
    {
        for(j=1; j<=n; j++)
            printf("%d ",a[i][j]);
        printf("\n");
    }
    getch();
}
```

```
4)#include<stdio.h>
#include<conio.h>
int temp[10],k=0;
void sort(int a[][10],int id[],int n)
{
    int i,j;
    for(i=1; i<=n; i++)
    {
        if(id[i]==0)
        {
            id[i]=-1;
            temp[++k]=i;
            for(j=1; j<=n; j++)
            {
                if(a[i][j]==1 && id[j]!=-1)
                    id[j]--;
            }
            i=0;
        }
    }
}
void main()
{
    int a[10][10],id[10],n,i,j;
    printf("\nEnter the n value:");
    scanf("%d",&n);
    for(i=1; i<=n; i++)
        id[i]=0;
    printf("\nEnter the graph data:\n");
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
        {
            scanf("%d",&a[i][j]);
            if(a[i][j]==1)
                id[j]++;
        }
    sort(a,id,n);
    if(k!=n)
        printf("\nTopological ordering not possible");
    else
    {
        printf("\nTopological ordering is:");
        for(i=1; i<=k; i++)
            printf("%d ",temp[i]);
    }
    getch();
}
```

6)

```
#include<stdio.h>
int w[10],p[10],n;
int max(int a,int b)
{
    return a>b?a:b;
}
int knap(int i,int m)
{
    if(i==n) return w[i]>m?0:p[i];
    if(w[i]>m) return knap(i+1,m);
    return max(knap(i+1,m),knap(i+1,m-w[i])+p[i]);
}
int main()
{
    int m,i,max_profit;
    printf("\nEnter the no. of objects:");
    scanf("%d",&n);
    printf("\nEnter the knapsack capacity:");
    scanf("%d",&m);
    printf("\nEnter profit followed by weight:\n");
    for(i=1; i<=n; i++)
        scanf("%d %d",&p[i],&w[i]);
    max_profit=knap(1,m);
    printf("\nMax profit=%d",max_profit);
    return 0;
}
```

```
7)#include <stdio.h>
#define MAX 50
int p[MAX], w[MAX], x[MAX];
double maxprofit;
int n, m, i;
void greedyKnapsack(int n, int w[], int p[], int m)
{
    double ratio[MAX];

// Calculate the ratio of profit to weight for each item
    for (i = 0; i < n; i++)
    {
        ratio[i] = (double)p[i] / w[i];
    }
// Sort items based on the ratio in non-increasing order
    for (i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (ratio[i] < ratio[j])
            {
                double temp = ratio[i];
                ratio[i] = ratio[j];
                ratio[j] = temp;

                int temp2 = w[i];
                w[i] = w[j];
                w[j] = temp2;

                temp2 = p[i];
                p[i] = p[j];
                p[j] = temp2;
            }
        }
    }
    int currentWeight = 0;
    maxprofit = 0.0;
// Fill the knapsack with items
    for (i = 0; i < n; i++)
    {
        if (currentWeight + w[i] <= m)
        {
            x[i] = 1; // Item i is selected
            currentWeight += w[i];
            maxprofit += p[i];
        }
        else
        {
// Fractional part of item i is selected
            x[i] = (m - currentWeight) / (double)w[i];
            maxprofit += x[i] * p[i];
            break;
        }
    }
    printf("Optimal solution for greedy method: %.1f\n", maxprofit);
    printf("Solution vector for greedy method: ");
    for (i = 0; i < n; i++)
        printf("%d\t", x[i]);
}
int main()
{
    printf("Enter the number of objects: ");
    scanf("%d", &n);
    printf("Enter the objects' weights: ");
    for (i = 0; i < n; i++)
        scanf("%d", &w[i]);
    printf("Enter the objects' profits: ");
    for (i = 0; i < n; i++)
        scanf("%d", &p[i]);
    printf("Enter the maximum capacity: ");
    scanf("%d", &m);
    greedyKnapsack(n, w, p, m);
    return 0;
}

9)#include<stdio.h>
#include<conio.h>
#include<time.h>
void mergeSort(int arr[], int left, int right);
void generateRandomArray(int arr[], int n);
void main()
{
long int arrayone[10000];
double d,start,end;
long int i,randomIndex,temp;
int m,count,j=0;
double timer[10],e,s,st[10],et[10];
int elements[10];
count=1;
while(count<=10)
{
printf("enter how many elements you want to generate\n");
printf("\n");
scanf("%d",&m);
start=clock();
for (i=0;i<m;i++)
    arrayone[i] = i;
for (i=0;i<m;i++)
 {
    temp=arrayone[i];
    randomIndex = rand() % m;
    arrayone[i] = arrayone[randomIndex];
    arrayone[randomIndex] = temp;
}
void mergeSort(int arr[], int left, int right);
printf("Array before sorting:");
printf("\n");
for(i=0;i<m;i++)
printf("%ld\t",arrayone[i]);
printf("\n");
void mergeSort(int arr[], int left, int right);
printf("Array after sorting:");
printf("\n");
for(i=0;i<m;i++)
```

```
printf("%ld\t",arrayone[i]);
printf("\n");
end=clock();
s=(double)start/CLOCKS_PER_SEC;
e=(double)end/CLOCKS_PER_SEC;
d=(double)(end-start)/CLOCKS_PER_SEC;
count++;
timer[j]=d;
st[j]=s;
et[j]=e;
elements[j]=m;
j++;
}
printf("Start_time|\tEnd_Time|\tNo_of_Elements||||Executiontime");
printf("\n");
for(i=0;i<10;i++)
{
printf("%lf\t%lf\t%d\t\t%lf",st[i],et[i],elements[i],timer[i]);
printf("\n");
}
getch();
}
// Function to generate random integers
void generateRandomArray(int arr[], int n)
{
   for (int i = 0; i < n; i++)
      arr[i] = rand() % 100000; // Generate random integers between 0 and 99999
}
void merge(int arr[], int left, int mid, int right)

{
   int i, j, k;
   int n1 = mid - left + 1;
   int n2 = right - mid;

   int *L = (int *)malloc(n1 * sizeof(int));
   int *R = (int *)malloc(n2 * sizeof(int));

   for (i = 0; i < n1; i++)
      L[i] = arr[left + i];
   for (j = 0; j < n2; j++)
      R[j] = arr[mid + 1 + j];

   i = 0;
```

```
    j = 0;
    k = left;

    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++;
    }

    free(L);
    free(R);
}
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);
    }
}
```

10)
```
#include<stdio.h>
#include<conio.h>
#include<time.h>
int partition(int arr[], int low, int high);
void quickSort(int arr[], int low, int high);
void swap(int* a, int* b);
void main()
{
long int arrayone[10000];
double d,start,end;
long int i,randomIndex,temp;
int m,count,j=0;
double timer[10],e,s,st[10],et[10];
int elements[10];
count=1;
while(count<=10)
{
printf("enter how many elements you want to generate\n");
printf("\n");
scanf("%d",&m);
start=clock();
for (i=0;i<m;i++)
    arrayone[i] = i;
for (i=0;i<m;i++)
 {
    temp=arrayone[i];
    randomIndex = rand() % m;
    arrayone[i] = arrayone[randomIndex];
    arrayone[randomIndex] = temp;
}
void quickSort(int arr[], int low, int high);
printf("Array before sorting:");
printf("\n");
for(i=0;i<m;i++)
printf("%ld\t",arrayone[i]);
printf("\n");
void quickSort(int arr[], int low, int high);
printf("Array after sorting:");
printf("\n");
for(i=0;i<m;i++)
```

```
printf("%ld\t",arrayone[i]);
printf("\n");
end=clock();
s=(double)start/CLOCKS_PER_SEC;
e=(double)end/CLOCKS_PER_SEC;
d=(double)(end-start)/CLOCKS_PER_SEC;
count++;
timer[j]=d;
st[j]=s;
et[j]=e;
elements[j]=m;
j++;
}
printf("Start_time|\tEnd_Time|\tNo_of_Elements||||Executiontime");
printf("\n");
for(i=0;i<10;i++)
{
printf("%lf\t%lf\t%d\t\t%lf",st[i],et[i],elements[i],timer[i]);
printf("\n");
}
getch();
int partition(int arr[], int low, int high)
{
   int pivot = arr[high]; // Pivot element
   int i = (low - 1); // Index of smaller element

   for (int j = low; j <= high - 1; j++)
   {
      if (arr[j] < pivot)
      {
         i++; // Increment index of smaller element
         swap(&arr[i], &arr[j]);
      }
   }
   swap(&arr[i + 1], &arr[high]);
   return (i + 1);
}

// Quick Sort function
void quickSort(int arr[], int low, int high)
{
   if (low < high)
   {
      int pi = partition(arr, low, high);
```

```
        // Recursively sort elements before and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}}
```

11)

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
void selectionsort(long int [],int);
void main()
{
long int arrayone[10000];
double d,start,end;
long int i,randomIndex,temp;
int m,count,j=0;
double timer[10],e,s,st[10],et[10];
int elements[10];
count=1;
while(count<=10)
{
printf("enter how many elements you want to generate\n");
printf("\n");
scanf("%d",&m);
start=clock();
for (i=0;i<m;i++)
    arrayone[i] = i;
for (i=0;i<m;i++)
 {
    temp=arrayone[i];
    randomIndex = rand() % m;
    arrayone[i] = arrayone[randomIndex];
    arrayone[randomIndex] = temp;
}
selectionsort(arrayone,m);
printf("Array before sorting:");
printf("\n");
for(i=0;i<m;i++)
printf("%ld\t",arrayone[i]);
printf("\n");
selectionsort(arrayone,m);
printf("Array after sorting:");
printf("\n");
for(i=0;i<m;i++)
printf("%ld\t",arrayone[i]);
printf("\n");
end=clock();
s=(double)start/CLOCKS_PER_SEC;
e=(double)end/CLOCKS_PER_SEC;
d=(double)(end-start)/CLOCKS_PER_SEC;
count++;
timer[j]=d;
st[j]=s;
et[j]=e;
elements[j]=m;
j++;
}
printf("Start_time|\tEnd_Time|\tNo_of_Elements||||Executiontime");
printf("\n");
for(i=0;i<10;i++)
{
printf("%lf\t%lf\t%d\t\t%lf",st[i],et[i],elements[i],timer[i]);
printf("\n");
}
getch();
}
void selectionsort(long int a[],int m)
{
int i,j,t,pos;
for(i=0;i<m-1;i++)
{
pos=i;
for(j=i+1;j<m;j++)
{
if(a[j]<a[pos])
pos=j;
}
t=a[i];
a[i]=a[pos];
a[pos]=t;
```

```
}
}
```