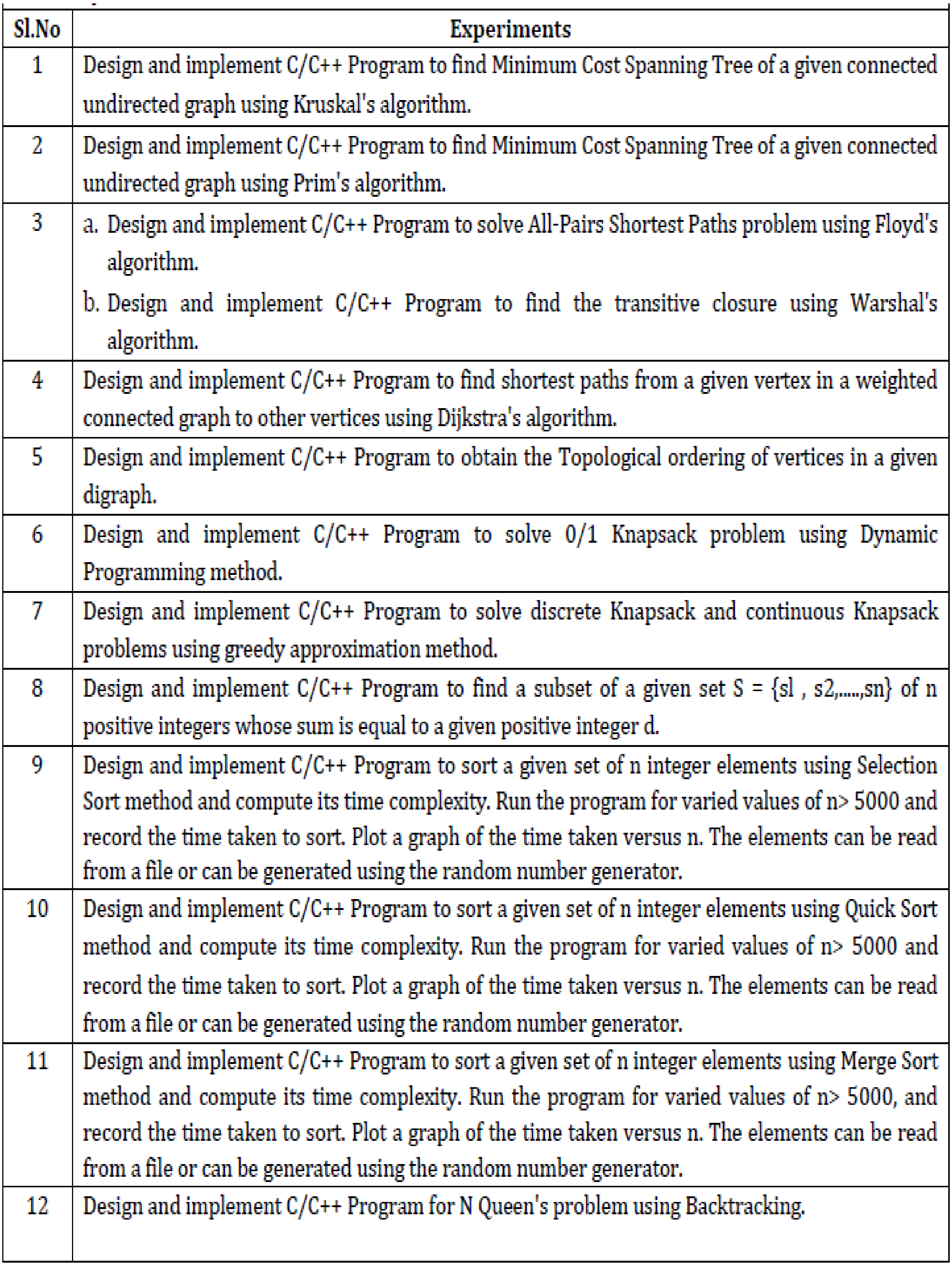
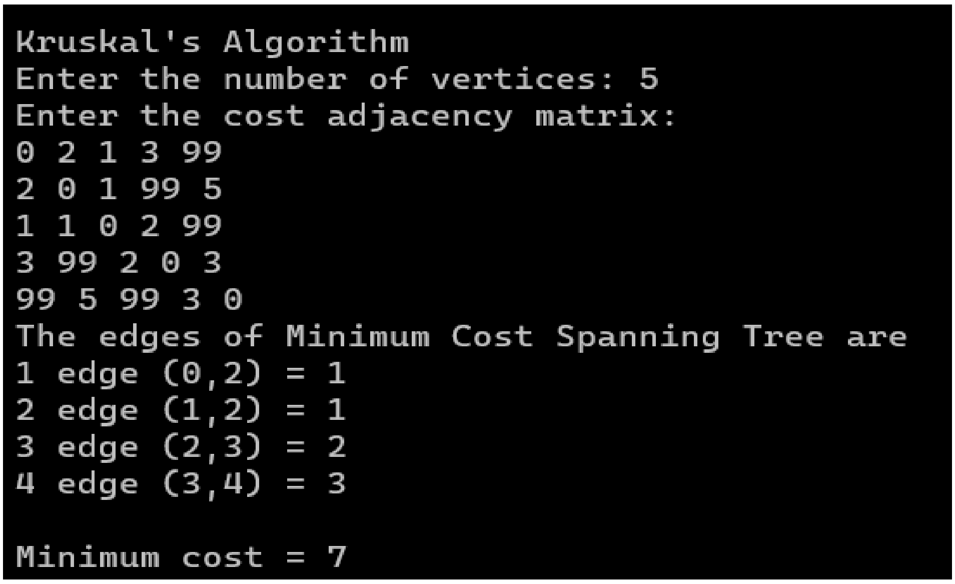
ADA LAB MANUAL



Sl.No	Experiments
1	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.
2	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.
3	a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm. b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.
4	Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.
5	Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph.
6	Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.
7	Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.
8	Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d.
9	Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.
10	Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.
11	Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.
12	Design and implement C/C++ Program for N Queen's problem using Backtracking.

1.KRUSKAL’S ALGORITHM

```
#include<stdio.h>
int parent[10],min_cost;
int find(int i)
  {
    if (parent[i]!= i)
     parent[i]=find(parent[i]);
     return parent[i];
  }

void main()
 {
 int    i, j, u, v, n, count = 1,cost[10][10],min,a,b;
 printf("\nKruskal's Algorithm\n");
 printf("Enter the number of vertices: ");
 scanf("%d", &n);
 printf("Enter the cost adjacency matrix:\n");
 for (i =0; i <n; i++)
 for (j =0; j < n; j++)
  scanf("%d", &cost[i][j]);
 for (i =0 ; i<n ;  i++)
  parent[i] = i;
 printf("The edges of Minimum Cost Spanning Tree are\n");
 while (count < n)
 {
    min = 99;
    for (i =0; i <n; i++)
      for (j =0; j < n; j++)
        if (cost[i][j] < min)
         {
           min = cost[i][j];
           u = i;
           v = j;
         }
     a=find(u);
     b=find(v);
    if(a!=b)
    {
     parent[b] = a;
     printf("%d edge (%d,%d) = %d\n", count++,u,v, min);
     min_cost = min_cost + min;
    }
   cost[u][v] = cost[v][u] =99;
 }
 printf("\nMinimum cost = %d\n", min_cost);
}
```

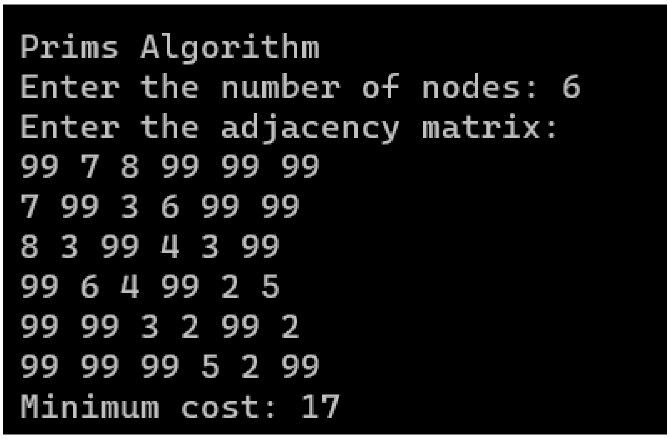


```
Kruskal's Algorithm
Enter the number of vertices: 5
Enter the cost adjacency matrix:
0 2 1 3 99
2 0 1 99 5
1 1 0 2 99
3 99 2 0 3
99 5 99 3 0
The edges of Minimum Cost Spanning Tree are
1 edge (0,2) = 1
2 edge (1,2) = 1
3 edge (2,3) = 2
4 edge (3,4) = 3

Minimum cost = 7
```

2.PRIMS ALGORITHM

```
#include <stdio.h>
int   visited[10] , mincost ;

void main()
{
            int    n, i, j,count =1,u,v,cost[10][10],min;
            printf("Prims Algorithm\n");
            printf("Enter the number of nodes: ");
            scanf("%d", &n);
            printf("Enter the adjacency matrix:\n");
            for (i = 0; i < n; i++)
            for (j = 0 ; j < n; j++)
            scanf("%d", &cost[i][j]);
            visited[0] = 1;
           printf("The edges of Minimum Cost Spanning Tree are\n");
            while (count < n)
              {
                min = 99;
                for (i = 0; i < n; i++)
                for (j = 0; j < n; j++)
                   if (cost[i][j] < min &&  i!=j  &&  visited[i] == 1  &&  visited[j] == 0)
                         {
                             min = cost[i][j];
                             u = i;
                             v = j;
                         }
                  printf("%d edge (%d,%d) = %d\n", count++, u, v, min);
                  mincost = mincost + min;
                  visited[v] =1;
            }
    printf("Minimum cost: %d\n", mincost);
}
```

OUTPUT

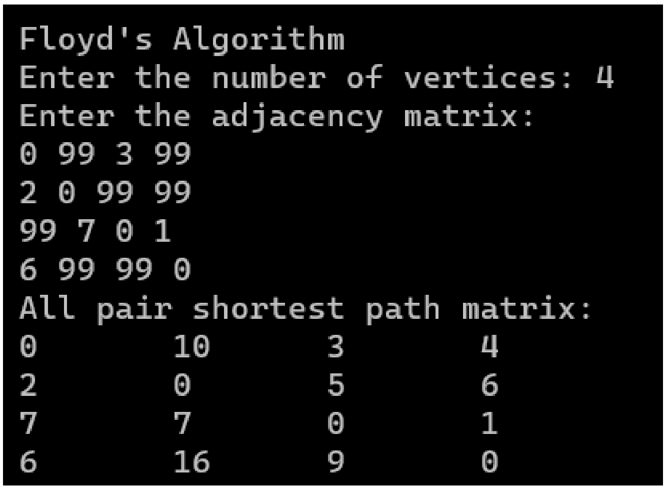


```
Prims Algorithm
Enter the number of nodes: 6
Enter the adjacency matrix:
99 7 8 99 99 99
7 99 3 6 99 99
8 3 99 4 3 99
99 6 4 99 2 5
99 99 3 2 99 2
99 99 99 5 2 99
Minimum cost: 17
```

3a FLOYD’S ALGORITHM

```
#include<stdio.h>
void main(){
int  d[10][10],i,j,k,n;
printf("\nFloyd's Algorithm\n");
printf("Enter the number of vertices");
scanf("%d",&n);
printf("Enter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
  scanf("%d",&d[i][j]);
for(k=0;k<n;k++)
for(i=0;i<n;i++)
for(j=0;j<n;j++)
  if( (d[i][k]+d[k][j]) < d[i][j] )
    d[i][j]= d[i][k]+d[k][j];
printf("All pair shortest path matrix:\n");
for(i=0;i<n;i++)
{
  for(j=0;j<n;j++)
   printf("%d\t",d[i][j]);
  printf("\n");
 }
}
```

OUTPUT

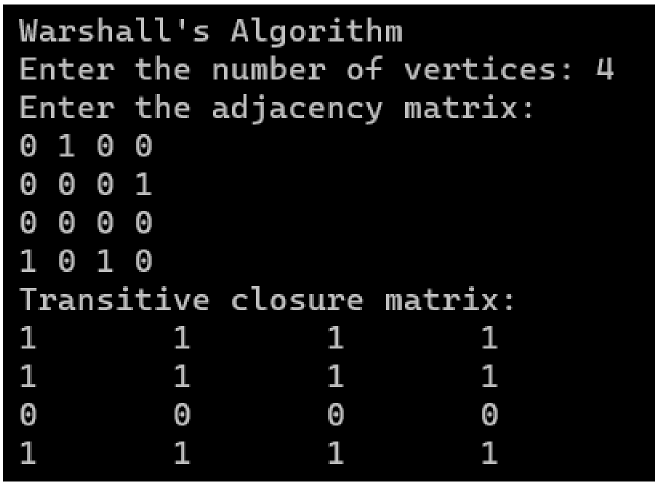


```
Floyd's Algorithm
Enter the number of vertices: 4
Enter the adjacency matrix:
0 99 3 99
2 0 99 99
99 7 0 1
6 99 99 0
All pair shortest path matrix:
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0
```

3b WARSHALL’S ALGORITHM

```
#include<stdio.h>
void main(){
int   d[10][10],i,j,k,n;
printf("\nWarshall's Algorithm\n");
printf("Enter the number of vertices: ");
scanf("%d",&n);
printf("Enter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
  scanf("%d",&d[i][j]);
for(k=0;k<n;k++)
for(i=0;i<n;i++)
for(j=0;j<n;j++)
  if(d[i][k]==1 && d[k][j]==1)
     d[i][j]=1;
printf("Transitive closure matrix:\n");
for(i=0;i<n;i++)
{
  for(j=0;j<n;j++)
   printf("%d\t",d[i][j]);
  printf("\n");
 }
}
```

OUTPUT

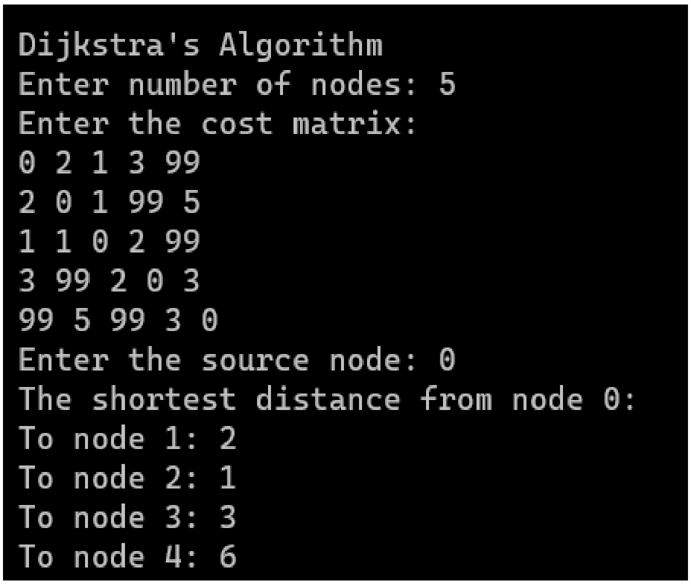


```
Warshall's Algorithm
Enter the number of vertices: 4
Enter the adjacency matrix:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
Transitive closure matrix:
1       1       1       1
1       1       1       1
0       0       0       0
1       1       1       1
```

4.DIJKSTRA’S ALGORITHM

```
#include <stdio.h>
int   visited[10] ;
void main()
 {
   int   n,src,i, j , cost[10][10],dist[10],min,u,count=1;
   printf("\n Dijkstra's Algorithm\n");
   printf("Enter number of nodes: ");
   scanf("%d", &n);
   printf("Enter the cost matrix:\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         scanf("%d", &cost[i][j]);
    for (i = 0; i < n; i++)
    dist[i] = 99;
   printf("Enter the source node: ");
   scanf("%d", &src);
   dist[src] = 0;
   while (count < n)
    {
      min = 99;
      for (i = 0; i < n; i++)
         if (dist[i] < min && visited[i]==0)
          {
            min = dist[i];
            u = i;
          }
       visited[u] = 1;
       count++;
      for (i = 0; i < n; i++)
         if (visited[i]==0 && (dist[u] + cost[u][i] < dist[i]))
          dist[i] = dist[u] + cost[u][i];
     }
   printf("The shortest distance from node %d:\n", src);
   for (i = 0; i < n; i++)
   printf("To node %d: %d\n", i, dist[i]);
  }
```

OUTPUT

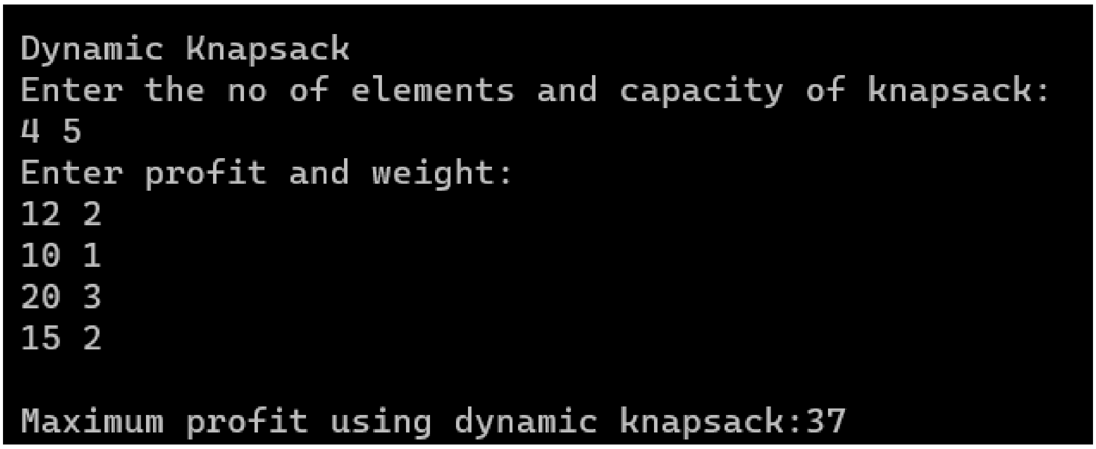


```
Dijkstra's Algorithm
Enter number of nodes: 5
Enter the cost matrix:
0 2 1 3 99
2 0 1 99 5
1 1 0 2 99
3 99 2 0 3
99 5 99 3 0
Enter the source node: 0
The shortest distance from node 0:
To node 1: 2
To node 2: 1
To node 3: 3
To node 4: 6
```

6.DYNAMIC KNAPSACK

```
#include<stdio.h>
void main()
{
   int n,m,i,j,p[10],w[10],v[20][20]={0};
   printf("\nDynamic Knapsack\n");
   printf("Enter the no of elements and capacity of knapsack:\n");
   scanf("%d%d",&n,&m);
   printf("Enter profit and weight:\n");
   for(i=1;i<=n;i++)
      scanf("%d %d",&p[i],&w[i])
   for(i=1;i<=n;i++)
    for(j=1;j<=m;j++)
     if(w[i]>j)
       v[i][j]=v[i-1][j];
     else
      if(v[i-1][j] > (p[i]+v[i-1][j-w[i]]))
        v[i][j]=v[i-1][j];
      else
        v[i][j]=p[i]+v[i-1][j-w[i]];

   printf("\nMaximum profit using dynamic knapsack:%d",v[n][m]);
}
```

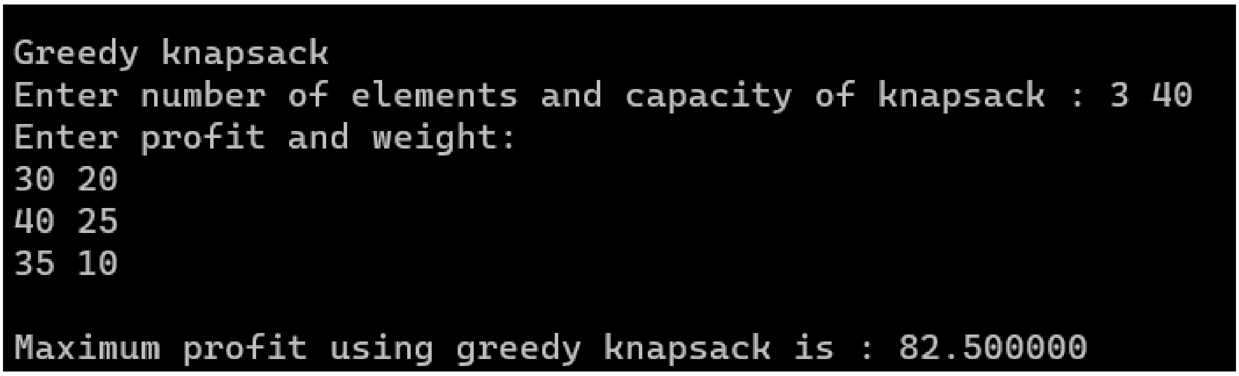
OUTPUT



```
Dynamic Knapsack
Enter the no of elements and capacity of knapsack:
4 5
Enter profit and weight:
12 2
10 1
20 3
15 2

Maximum profit using dynamic knapsack:37
```

7.GREEDY KNAPSACK

```
#include<stdlib.h>
#include<stdio.h>
void main()
{
        int n,m,i,j;
        float r[10],profit=0,w[10],p[10],temp;
        printf("\nGreedy knapsack\n");
        printf("Enter number of elements and capacity of knapsack : ");
        scanf("%d%d",&n,&m);
        printf("Enter profit and weight:\n");
        for(i=0;i<n;i++)
        {
                scanf("%f%f",&p[i],&w[i]);
                r[i]=p[i]/w[i];
        }
        for(i=0;i<n-1;i++)
                for(j=0;j<n-1-i;j++)
                        if(r[j]<r[j+1])
                        {
                                temp=r[j], r[j]=r[j+1], r[j+1]=temp;
                                temp=w[j], w[j]=w[j+1], w[j+1]=temp;
                                temp=p[j], p[j]=p[j+1], p[j+1]=temp;
                        }
        for(i=0;i<n;i++)
        {
                if(w[i]<m)
                {       profit=profit+p[i];
                        m=m-w[i];
                }
                else
                {       profit=profit + m/w[i]*p[i];
                        break;
                }
        }

    printf("\nMaximum profit using greedy knapsack is : %f ",profit);
}
```

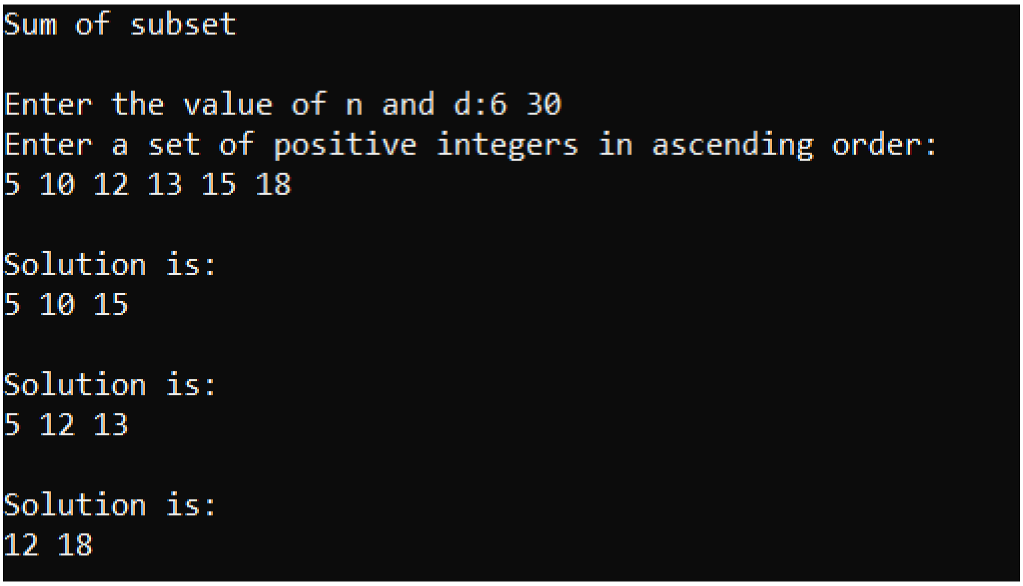


```
Greedy knapsack
Enter number of elements and capacity of knapsack : 3 40
Enter profit and weight:
30 20
40 25
35 10

Maximum profit using greedy knapsack is : 82.500000
```

8.SUM OF SUBSET

```
#include<stdio.h>
int w[10],x[10],d,i,n,sum=0;
void sumofsub(int cs ,int k,int sum)
 {
   x[k]=1;
   if((cs+w[k])==d)
   {
     printf("\nSolution is:\n");
      for(i=0;i<=k;i++)
       if(x[i]==1)
      printf("%d ",w[i]);
       printf("\n");
    }
else if(cs+w[k]+w[k+1]<=d)
  sumofsub(cs+w[k],k+1,sum-w[k]);
if((cs+sum-w[k]>=d) && (cs+w[k+1]<=d))
   {
     x[k]=0;
     sumofsub(cs,k+1,sum-w[k]);
    }
}
void main()
 {
printf("\nSum of subset\n");
printf("\nEnter the value of n and d:");
scanf("%d%d",&n,&d);
printf("Enter a set of positive integers in ascending order:\n");
for(i=0;i<n;i++)
 {  scanf("%d",&w[i]);
    sum=sum+w[i];
  }
sumofsub(0,0,sum);
}
```

OUTPUT



```
Sum of subset

Enter the value of n and d:6 30
Enter a set of positive integers in ascending order:
5 10 12 13 15 18

Solution is:
5 10 15

Solution is:
5 12 13

Solution is:
12 18
```

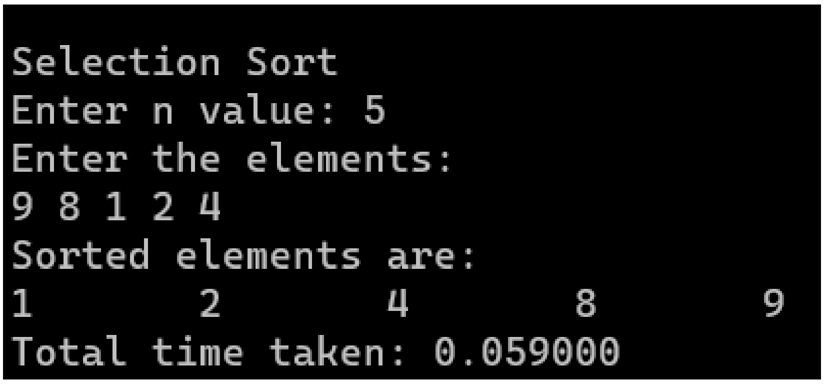
NOTE:

cs=currentsum
sum=totalsum
k=index of integer

9. Selection Sort

```
#include<stdio.h>
#include<time.h>
void main()
{
        int i,j,n,min,temp,a[10];
        clock_t start,end;
        double time;
        printf("\nSelection Sort\n");
        printf("Enter the number of elements: ");
        scanf("%d",&n);
        printf("Enter the elements:\n");
        for(i=0;i<n;i++)
           scanf("%d",&a[i]);
        start=clock();
        for(i=0;i<n-1;i++)
         {
          usleep(1000);
          min=i;
          for(j=i+1;j<n;j++)
            if(a[j]<a[min])
                min=j;
            temp=a[i];
            a[i]=a[min];
            a[min]=temp;
         }
        end=clock();
        time=((double)(end-start))/CLOCKS_PER_SEC;
        printf("Sorted elements are:\n");
        for(i=0;i<n;i++)
          printf("%d\t",a[i]);
        printf("\nTotal time taken: %f",time);
}
```

OUTPUT



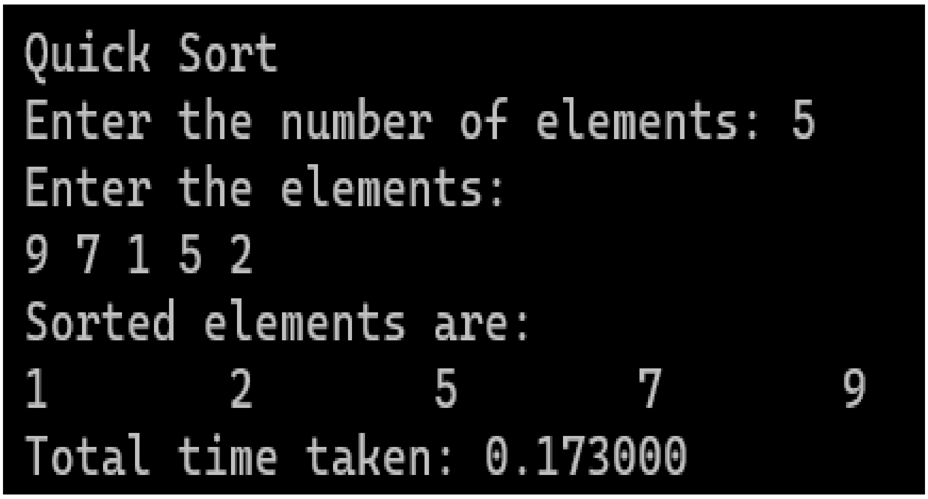
```
Selection Sort
Enter n value: 5
Enter the elements:
9 8 1 2 4
Sorted elements are:
1       2       4       8       9
Total time taken: 0.059000
```

10.Quick Sort

```
#include<stdio.h>
#include<time.h>
int part(int a [20],int low,int high)
{
	int i,j,temp,key;
	key=a[low],i=low+1,j=high;
	while(1)
	{
	while (i<high && key>=a[i])i++;
	while(key<a[j])j--;
	if(i<j)
	{
		temp=a[i];
		a[i]=a[j];
		a[j]=temp;
	}
	else
	{
		temp=a[low];
		a[low]=a[j];
		a[j]=temp;
		return j;
	}
	}
}


void quick_sort(int a[20],int low,int high)
{
  int mid;
  usleep(1000);
  if(low<=high)
  {
  mid=part(a,low,high);
  quick_sort(a,low,mid-1);
  quick_sort(a,mid+1,high);
  }
}
```

```
void main()
{
        int i,n,a[20];
        clock_t start,end;
        double time;
        printf("\nQuick Sort\n");
        printf("Enter the number of elements: ");
        scanf("%d",&n);
        printf("Enter the elements:\n");
        for(i=0;i<n;i++)
        scanf("%d",&a[i]);
        start=clock();
        quick_sort(a,0,n-1);
        end=clock();
        time=((double)(end-start))/CLOCKS_PER_SEC;
        printf("Sorted elements are:\n");
        for(i=0;i<n;i++)
        printf("%d\t",a[i]);
        printf("\nTotal time taken: %f",time);
}
```

OUTPUT



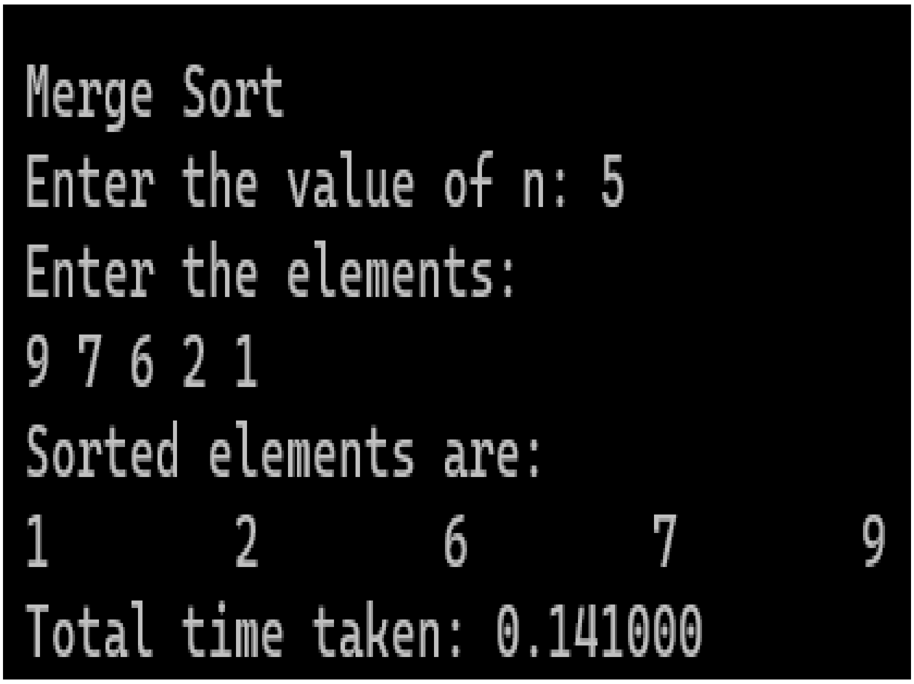
```
Quick Sort
Enter the number of elements: 5
Enter the elements:
9 7 1 5 2
Sorted elements are:
1       2       5       7       9
Total time taken: 0.173000
```

11.Merge Sort

```
#include<stdio.h>
#include<time.h>
void merging(int a[],int low,int mid,int high)
{
        int i=low, j=mid+1,k=low,c[20];
        while(i<=mid && j<=high)
        if(a[i]<a[j])
                c[k++]=a[i++];
        else
                c[k++]=a[j++];
        while(i<=mid)
                c[k++]=a[i++];
        while(j<=high)
                c[k++]=a[j++];
        while(low<=high)
        {
                a[low]=c[low];
                low++;
        }
}


void merge_sort(int a[],int low, int high)
{
        int mid;
        usleep(1000);
        if(low<high)
        {
                mid=(low+high)/2;
                merge_sort(a,low,mid);
                merge_sort(a,mid+1,high);
                merging(a,low,mid,high);
        }
}
```

```
void main()
{
	int i,n,a[20];
	clock_t start,end;
	double time;
	printf("\nMerge Sort\n");
	printf("Enter the number of elements: ");
	scanf("%d",&n);
	printf("Enter the elements:\n");
	for(i=0;i<n;i++)
	scanf("%d",&a[i]);
	start=clock();
	merge_sort(a,0,n-1);
	end=clock();
	time=((double)(end-start))/CLOCKS_PER_SEC;
	printf("Sorted elements are:\n");
	for(i=0;i<n;i++)
	printf("%d\t",a[i]);
	printf("\nTotal time taken: %f",time);
}
```

OUTPUT



```
Merge Sort
Enter the value of n: 5
Enter the elements:
9 7 6 2 1
Sorted elements are:
1       2       6       7       9
Total time taken: 0.141000
```