Program 1 :	Write a program to search an element in an array using Linear Search method. Find the time required to search an element.

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>

int linearSearch(int a[],int n,int key)
{
        int i;

        for(i=0;i<n;i++)
        {
            if (key == a[i])
                return i;
        }
        return -1;        //Unsuccessful Search - It returns -1 if key is not found
}

void main()
{
        char ch;
        int a[100], n, key, i, res;
        clock_t st,et;

        printf("Enter the number of elements in the array : \n");
        scanf("%d",&n);

        printf("Enter the elements of the array :\n");
        for(i=0;i<n;i++)
            scanf("%d",&a[i]);

        printf("Enter the key  element to search : \n");
        scanf("%d",&key);

        st=clock();                // Record Start Time
        res=linearSearch(a,n,key);
        et=clock();                // Record End time
        double time_taken = (((double) (et - st)) / CLOCKS_PER_SEC)*1000;

        if(res==-1)
        {
            printf("The search element is not found\n");
            printf("The Execution Time is = %.0f Milli Seconds",time_taken);
            exit(0);
        }
        else
            printf("The search element is found at position %d\n",res+1);
        printf("The Execution Time is = %.0f Milli Seconds",time_taken);
}
```

Output

```
Enter the number of elements in the array :
5
Enter the elements of the array :
10 20 30 40 50
Enter the key element to search :
50
The search element is found at position 5
The Execution Time is = 0 Milli Seconds
```

Program 2 : **Write a program to search an element in an array using Binary Search method. Find the time required to search an element.**

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>

int binarySearch(int a[],int key,int n,int first,int last)
{
    int mid, i, j, temp;

    if (last<first)
        return -1;

    for(i=0;i<=n-2;i++)
    {
        for(j=0;j<=n-2-i;j++)
        {
            if(a[j+1] < a[j])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }

    while(first<=last)
    {
        mid=(first+last)/2;
        if(key==a[mid])
            return mid+1;
        else if(key<a[mid])
            last=mid-1;
        else
            first=mid+1;
    }
    return -1;
}
```

```
void main()
{
        char ch;
        int a[100],n, key, i, res, first, last;

        clock_t st,et;

        printf("Enter the number of elements in the array : \n");
        scanf("%d",&n);

        printf("Enter the elements of the array in  :\n");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);

        printf("Enter the key  element to search : \n");
        scanf("%d",&key);

        first=0;
        last=n-1;

        st=clock();                         // Record Start Time

        res=binarySearch(a,key,n,first,last);
        printf("The sorted array is : ");
        for(i=0;i<n;i++)
            printf(" %d ",a[i]);

        et=clock();                         // Record End time
        double time_taken = (((double) (et - st)) / CLOCKS_PER_SEC)*1000;

        if(res==-1)
        {
            printf("\nThe search element is not found\n");
            printf("The Execution Time is = %.0f Milli Seconds",time_taken);
            exit(0);
        }
        else
                printf("\nThe search element is found at position %d\n",res);
        printf("The Execution Time is = %.0f Milli Seconds",time_taken);
}
```

Output

```
Enter the number of elements in the array :
6
Enter the elements of the array in :
60 40 30 20 10 50
Enter the key element to search :
10
The sorted array is : 10 20 30 40 50 60
The search element is found at position 1
The Execution Time is = 2 Milli Seconds
```

Program 3 : **Write a program to search an element in an array using Recursive Binary Search method. Find the time required to search an element.**

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>

int RBinarySearch(int a[],int key,int n,int first,int last)
{
        int mid,i,j,temp;

        if(last<first)
            return -1;

        for(i=0;i<=n-2;i++)
        {
            for(j=0;j<=n-2-i;j++)
            {
                if(a[j+1] < a[j])
                {
                    temp=a[j];
                    a[j]=a[j+1];
                    a[j+1]=temp;
                }
            }
        }
        mid=(first+last)/2;

        if(key==a[mid])
            return mid;
        else if(key<a[mid])
            return RBinarySearch(a,key,n,first,mid-1);
        else
            return RBinarySearch(a,key,n,mid+1,last);
}

void main()
{
        char ch;
        int a[100],n,key,i,res, first, last;
        clock_t st,et;
        printf("Enter the number of elements in the array : \n");
        scanf("%d",&n);

        printf("Enter the elements of the array :\n");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);

        printf("Enter the key  element to search : \n");
        scanf("%d",&key);
        first=0;
        last=n-1;
```

```
    st=clock();                 // Record Start Time

    res=RBinarySearch(a,key,n,first,last);
    printf("The sorted array is : ");
    for(i=0;i<n;i++)
        printf(" %d ",a[i]);

    et=clock();                 // Record End time
    double time_taken = (((double) (et - st)) / CLOCKS_PER_SEC)*1000;

    if(res==-1)
    {
        printf("\nThe search element is not found\n");
        printf("The Execution Time is = %.0f Milli Seconds",time_taken);
        exit(0);
    }
    else
        printf("\nThe search element is found at position %d\n",res+1);
    printf("The Execution Time is = %.0f Milli Seconds",time_taken);
}
```

Output

```
Enter the number of elements in the array :
6
Enter the elements of the array :
90 70 50 80 40 30
Enter the key element to search :
40
The sorted array is : 30 40 50 70 80 90
The search element is found at position 2
The Execution Time is =  1 Milli Seconds
```

Program 4 : **Write a program to sort an elements in an array using Quick Sort. Find the time required to sort the elements.**

```
#include <stdio.h>
#include <conio.h>
#include <time.h>

void quicksort(int A[10], int low, int high)
{
    int j;
    if (low < high)
    {
        j=partition(A,low,high);
        quicksort(A, low, j - 1);
        quicksort(A, j + 1, high);
    }
}
```

```
int partition(int A[10], int low, int high)
{
        int pivot, j, temp, i;
        pivot = low;
        i = low;
        j = high;
        while (i< j)
        {
            while(i<high && A[i]<=A[pivot])
                i++;
            while (A[j]>A[pivot])
                j--;
            if (i< j)
            {
                temp = A[i];
                A[i] = A[j];
                A[j] = temp;
            }
        }

        temp = A[pivot];
        A[pivot] = A[j];
        A[j] = temp;
        return j;
}


void main()
{
        inti, n, A[10];
        clock_t st, et;

        printf("Enter the  number of elements of array : \n");
        scanf("%d", &n);

        printf("Enter the elements of the array : \n");
        for (i = 0; i< n; i++)
            scanf("%d", &A[i]);

        st = clock();
        quicksort(A, 0, n-1);
        et = clock();

        double time_taken = (((double) (et - st)) / CLOCKS_PER_SEC)*1000;

        printf("Sorted list of elements :");
        for (i = 0; i< n; i++)
            printf(" %d ", A[i]);

        printf("The Execution Time is = %.0f Milli Seconds",time_taken);
        getch();
}
```

Output

```
Enter the number of elements of array :
5
Enter the elements of the array :
30 20 10 50 40
Sorted list of elements : 10 20 30 40 50
The Execution Time is = 0 in milli seconds
```

Program 5 : **Write a program to sort an elements in an array using Merge Sort. Find the time required to sort the elements.**

```
#include <stdio.h>
#include <conio.h>
#include <time.h>

void Merge(int A[10],int low,int mid,int high)
{
        int i,j,k,c[10];
        i=low;
        k=low;
        j=mid+1;
        while((i<=mid)&&(j<=high))
        {
            if(A[i]<=A[j])
            {
                c[k]=A[i];
                i++;
                k++;
            }
            else
            {
                c[k]=A[j];
                j++;
                k++;
            }
        }

        while(i<=mid)
        {
            c[k]=A[i];
            k++;
            i++;
        }
        while(j<=high)
        {
            c[k]=A[j];
            k++;
            j++;
        }

        for(i=low; i<=k-1; i++)
            A[i]=c[i];
}
```

```
void MergeSort(int A[10],int low,int high)
{
        int mid;

        if(low<high)
        {
            mid=(low+high)/2;
            MergeSort(A,low,mid);
            MergeSort(A,mid+1,high);
            Merge(A,low,mid,high);
        }
}


void main()
{
        inti, n, A[10];
        clock_t st, et;

        printf("Enter the  number of elements of array : \n");
        scanf("%d", &n);

        printf("Enter the elements of the array : \n");
        for (i = 0; i< n; i++)
            scanf("%d", &A[i]);

        st = clock();
        MergeSort(A, 0, n-1);
        et = clock();
        double time_taken = (((double) (et - st)) / CLOCKS_PER_SEC)*1000;

        printf("Sorted list of elements :");
        for (i = 0; i< n; i++)
            printf("  %d  ", A[i]);

        printf("The Execution Time is = %.0f Milli Seconds",time_taken);
        getch();
}
```

Output

```
Enter the number of elements of array :
5
Enter the elements of the array :
28 9 5 44 12
Sorted list of elements : 5 9 12 28 44
The Execution Time is =0 in Milli Seconds
```

Program 6 : From a given vertex in a weighted connected graph find shortest paths to other vertices using Dijikstras Algorithm.

```
/* Shortest path using dijikstras algorithm */
#include<stdio.h>

void Dijikstra(int n,int v, int cost[10][10], int dist[])
{
        int count,u,i,w,flag[10],min;
        for(i=1;i<=n;i++)
        {
            flag[i]=0;
            dist[i]=cost[v][i];
        }
        flag[v]=1;
        dist[v]=1;
        count=2;

        while(count<=n)
        {
            min=9999;
            for(w=1; w<=n; w++)
                if(dist[w] < min && !flag[w])
                    min=dist[w], u=w;

            flag[u]=1;
            count++;

            for(w=1; w<=n; w++)
                if((dist[u]+cost[u][w] < dist[w]) && !flag[w])
                    dist[w] = dist[u]+ cost[u][w];
        }
}

void main()
{
        int n,v,cost[10][10],dist[10],i,j;
        printf("\n Enter the no of nodes:");
        scanf("%d",&n);

        printf("\n Enter the cost 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]=9999;
            }
```

```
        printf("Enter the source vertex: ");
        scanf("%d",&v);

        Dijikstra(n,v,cost,dist);

        printf("\n Shortest path from \n");
        for(j=1; j<=n; j++)
            if(j!=v)
                printf("\n%d->%d =%d",v,j,dist[j]);
        getch();
}
```

Output

```
Enter the no of nodes : 5

Enter the cost matrix :
    0     60    100     0     10
    0      0      0    50      0
    0      0      0     0     20
    0      0     20     0      0
    0      0      0     5      0
Enter the source vertex : 1.

Shortest path from

1->2 = 60
1->3 = 35
1->4 = 15
1->5 = 10
```

Program 7 : **Find minimum cost spanning tree of a given undirected graph using Kruskal's algorithm.**

```
/* Minimum cost spanning tree using Kruskal's algorithm */
#include<stdio.h>

int parent[10], min, no_of_edges=1,mincost=0,cost[10][10];
int a,b,i,j,u,v,n;

main()
{
        printf("\n Enter the no of vertices :");
        scanf("%d", &n);

        printf("\n Enter the adjacency matrix : ");
        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;
        }
    while(no_of_edges < n)
    {
        for(i=1, min=999; i<=n; i++)
            for(j=1; j<=n; j++)
                if(cost[i][j] < min)
                {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
                }
        while(parent[u])
            u=parent[u];
        while(parent[v])
            v=parent[v];

        if(u!=v)
        {
            printf("\n %d \t Edge \t(%d,%d) = %d", no_of_edges,a,b,min);
            mincost += min;
            parent[v] = u;
            no_of_edges++;
        }
        cost[a][b] = cost[b][a] = 999;
    }
    printf("\n\t Minimum cost = %d", mincost);
}
```

Output

```
Enter the no of vertices : 5

Enter the adjacency matrix :
 0   11   9   7   8
11    0  15  14  13
 9   15   0  12  14
 7   14  12   0   6
 8   13  14   6   0

 1    Edge(4,5) = 6
 2    Edge(1,4) = 7
 3    Edge(1,3) = 9
 4    Edge(1,2) = 11
      Minimum cost = 33
```

Program 8 : **Find minimum cost spanning tree of a given undirected graph using Prims algorithm**

```
/* Minimum cost spanning tree using prims algorithm */
#include<stdio.h>
int a,b,u,v,n,i,j,no_of_edges=1;
int visited[10], min, mincost=0, cost[10][10];

void main()
{
        printf("\n Enter the no of vertices : ");
        scanf("%d",&n);
        printf("\n Enter the adjacency matrix :");

        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;
            }
        for(i=2;i<=n;i++)
            visited[i]=0;

        printf("\n The edges of spanning tree are :\n");
        visited[1]=1;

        while(no_of_edges < n)
        {
            for(i=1, min=999; i<=n; i++)
                for(j=1; j<=n ;j++)
                    if(cost[i][j] < min)
                        if(visited[i]==0)
                            continue;
                        else
                        {
                            min = cost[i][j];
                            a=u=i;
                            b=v=j;
                        }
                    if(visited[u]==0 || visited [v]==0)
                    {
                        printf("\n%d\t Edge \t(%d,%d)=%d",no_of_edges,a,b,min);
                        mincost += min;
                        visited[b]=1;
                        no_of_edges++;
                    }
                cost[a][b]=cost[b][a]=999;
        }

        printf("\n\t Minimum cost = %d\n",mincost);
        getch();
}
```

Output

```
Enter the no of vertices and graph data : 4
   0     2     9     1
   2     5     1     11
   7     4     1     22
   11    4     7     2

The edges of spanning tree are:
   1                Edge(1,4) = 1
   2                Edge(1,2) = 2
   3                Edge(2,3) = 1
                    Minimum cost = 4
```

Program 9 : **Implement all pairs shortest paths problem using Floyd's algorithm.**

```
/* All pair shortest paths problem using Floyd's algorithm */
#include<stdio.h>

int c[5][5], n, i, j, k;

void floyd()
{
        for(k=1;k<=n;k++)
            for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                    if((c[i][k] + c[k][j]) < c[i][j])
                        c[i][j] = c[i][k] + c[k][j];
}
void main()
{
        printf("Enter the number of the verticies : \n");
        scanf("%d",&n);

        printf("\n Enter the adjacency matrix : \n");
        for(i=1; i<=n; i++)
                for(j=1; j<=n; j++)
            {
                scanf("%d",&c[i][j]);
                if(c[i][j]==0)
                        c[i][j]=999;
            }
```

```
    floyd();

    for(i=1;i<=n;i++)
        c[i][i]=0;
    printf("\n The all pair shortest path : \n");
    for(i=1;i<=n; i++)
    {
        for(j=1; j<=n; j++)
            printf("%6d",c[i][j]);
        printf("\n");
    }
    getch();
}
```

Output

```
Enter the number of verticies: 4

    Enter the adjacency matrix:
     0     10     0     40
     0      0     0     20
    50      0     0      0
     0      0    60      0

    The all pair shortest path:
     0     10    90     30
   130      0    80     20
    50     60     0     80
   110    120    60      0
```

Program 10: **Implement 0/1 knapsack problem using dynamic programming.**

```
/* 0/1 knapsack problem using dynamic programming */
#include<stdio.h>

int i,j,n,capacity,w[50],p[50],maxprofit;

int maximum(int x,int y)
{
    if(x>y)
        return x;
    else
        return y;
}
```

```
knapsack(int i,int c)
{
        if(i==n)
            return((c<w[n])? 0 : p[n]);

        if( c< w[i])
            return knapsack(i+1,c);
        return maximum(knapsack(i+1,c), knapsack(i+1,c-w[i])+p[i]);
}

void main()
{
        printf("\n Enter the no of objects :");
        scanf("%d",&n);

        printf("\n Enter the weights : ");
        for(i=0;i<n;i++)
            scanf("%d",&w[i]);

        printf("\n Enter the profits associated : ");
        for(i=0;i<n;i++)
            scanf("%d",&p[i]);

        printf("\n Enter the capacity : ");
        scanf("%d",&capacity);

        maxprofit=knapsack(0, capacity);
        printf("\n Maximum profit = %d", maxprofit);
        getch();
}
```

Output

```
Enter the number of objects :3

Enter the weights :
100
14
10

Enter the profits associated:
20
18
15

Enter the capacity
116

Maximum profit=38
```

Program 11: **Check whether a given graph is connected or not connected using DFS method.**

```
/* Check graph for connected using DFS */
#include<stdio.h>

int i,j,a[20][20],visited[20],n,label=0,k,I[10],v,w;

void initiate()
{
        for(i = 1; i<=n; i++)
            visited[i]=0,I[i]=0;

        for(i = 1; i <= n; i++)
            for(j = 1; j <= n; j++)
                a[i][j]=0;
}

void dfs(int v)
{
        visited[v]=1;
        I[v]=label;
        for(i=1; i<=n; i++)
            if(a[v][i] && visited[i]==0)
                dfs(i);
}

void component_label()
{
        int i;
        for(i=1; i<=n; i++)
        {
            if(I[i] == 0)
            {
                label++;
                for(k=1; k<=n; k++)
                visited [i]=0;
                dfs(i);
            }
        }
        if(label > 1)
            printf("\n Disconnected Graph.");
        else
            printf("\n Connected Graph");

        printf("\n No of connected elements : %d",label);
        for(i=1;i<=n;i++)
            printf("\n Vertex %d belongs to component %d",i,I[i]);
}
```

```
void main()
{
        char ch;
        printf("\n Enter the no vertices: ");
        scanf("%d",&n);

        initiate();
        do
        {
            printf("\n Enter the adjacent vertices: ");
            scanf("%d %d", &v,&w);
            if( v<0 || v>n || w<0 || w>n )
                printf("\n Invalid vertex .");
            else
            {
                a[v][w]=1;
                a[w][v]=1;
            }

            printf("\n Do you want to continue :");
            fflush(stdin);
            scanf("%c",&ch);

        }while(ch=='y' || ch=='y');

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

        for(i = 1;i <= n; i++)
        {
            for(j = 1; j <= n; j++)
                printf("%3d",a[i][j]);
            printf("\n");
        }

        component_label();
        getch();
}
```

Output

```
Enter the no vertices : 4

Enter the adjacent vertices:1  3
Do you want to continue : Y

Enter the adjacent vertices : 3 4
Do you want to continue : Y
```

```
Enter the adjacent vertices : 4 2
Do you want to continue : N

 The adjacency matrix is :
    0       0       1       0
    0       0       0       1
    1       0       0       1
    0       1       1       0
Connected Graph
No of connected elements : 1
Vertex 1 belongs to component 1
Vertex 2 belongs to component 1
Vertex 3 belongs to component 1
Vertex 4 belongs to component 1
```

Program 12: **Print all the nodes reachable from a given starting node in a given digraph using breadth first search**

```
#include<stdio.h>
#include<conio.h>

int q[10],f=-1,r=-1;

void qins(int x)
{
        if(f==-1 && r==-1)
        {
            f++;
            q[++r]=x;
        }
        else
            q[++r]=x;
}

int qdel()
{
        int x=q[f];
        if(f==r)
            f=r=-1;
        else
            f++;
        return x;
}
```

```
void main()
{
        int u,s[10]={0},src,n,a[10][10],i,j;

        printf("\n Enter no of vertices : \n");
        scanf("%d",&n);

        printf("\n Enter the adjacency matrix : \n)");
        for(i=1; i<=n;i++)
            for(j=1;j<=n;j++)
                scanf("%d",&a[i][j]);

        printf("\n Enter the source vertex   :");
        scanf("%d",&src);

        s[src]=1;

        qins(src);

        while(f!=-1 && r!=-1)
        {
            u=qdel();
            printf(" %d ",u);
            for(i=1;i<=n;i++)
                if(a[u][i]==1 &&s[i]==0)
                {
                    s[i]=1;
                    qins(i);
                }
        }

        printf("\n The nodes that are reachable from %d are :\n",src);
        for(i=1;i<=n;i++)
            if(s[i])
                printf("%d      \n",i);

        getch();
}
```

Output

```
Enter no of vertices :
5

Enter the adjacency matrix :
0 1 0 1 0
0 0 0 1 1
1 0 0 1 0
0 0 0 0 1
0 0 0 0 0
Enter the source vertex : 1
1 2 4 5
The nodes that are reachable from 1 are:
1
2
4
5
```

Program 13 : **Print all the nodes reachable from a given starting node in a given digraph using depth first search method.**

```
/* Check for nodes reachable using DFS */
#include<stdio.h>
#include<conio.h>

int s[10]={0};

void dfs(int n,int a[10][10],int u)
{
        int v;
        s[u]=1;
        printf(" %d ",u);
        for(v=1;v<=n;v++)
            if(a[u][v]==1 && s[v]==0)
                dfs(n,a,v);
}

void main()
{
        int a[10][10],n,src,i,j;
        printf("\n Enter the number of vertices(n) \n");
        scanf("%d",&n);
        printf("\n Enter the adjacency matrix\n");
```

```
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            scanf("%d",&a[i][j]);
    printf("\n Enter the source vertex ");
    scanf("%d",&src);
    if(src<=n)
    {
        dfs(n,a,src);
        printf("\n The nodes which are reachable from %d are :\n",src);
        for(i=1;i<=n;i++)
        if (s[i])
            printf(" %d ",i);
    }
    else
        printf("\n invalid source and try again");
}
```

Output

```
Enter the number of vertices(n)
4

Enter the adjacency matrix
    0    1    1    1
    1    0    1    0
    1    1    0    1
    1    0    1    0

Enter the source vertex 4
    4    1    2    3
The nodes which are reachable from 4 are :
    1    2    3    4
```

Program 14 : Implement N queens problem using backtracking

```
#include<stdio.h>
#include<stdlib.h>
int s[50][50];
void display(int m[],int n)
{
    register int i,j;
    for(i=0;i<n;i++)
        s[i][m[i]]=1;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            if(s[i][j])
                printf(" Q ");
            else
                printf(" x ");
        printf("\n");
    }
    exit(0);
}
```

```
int place( int m[], int k)
{
        int i;
        for(i=0; i<k; i++)
            if(m[i]==m[k] || (abs(m[i]-m[k]) == abs(i-k)))
                return 0;
        return 1;
}

void main()
{
        int m[25],n,k;

        printf("\n Enter the no of queens : ");
        scanf("%d",&n);

        printf("\n The solution to queens problem is \n");
        n--;
        for(m[0]=0, k=0; k>=0; m[k] += 1)
        {
            while(m[k] <= n && !place(m,k))
                    m[k] += 1;
            if(m[k] <= n)
                if(k==n)
                    display(m,n+1);
                else
                    k++, m[k] =-1;
            else
                k--;
        }
        getch();
}
```

Output

```
Enter the no of queens : 8
The solution to queens problem is
Q  x  x  x  x  x  x  x
x  x  x  x  Q  x  x  x
x  x  x  x  x  x  x  Q
x  x  x  x  x  Q  x  x
x  x  Q  x  x  x  x  x
x  x  x  x  x  x  Q  x
x  Q  x  x  x  x  x  x
x  x  x  Q  x  x  x  x

Enter the no of queens : 4
The solution to queens problem is
x  Q  x  x
x  x  x  Q
Q  x  x  x
x  x  Q  x
```

Program 15: Find a subset of a given set S =(S1,S2,.....,Sn) of N positive integers whose sum is equal to a given positive integer. For Example: If S = {1,2,5,6,8} and D = 9. There are two solutions (1,2,6) and (1,8). A suitable message is to be displayed if the given problem instance does not does not have a solution.

```
/* Sum of subset */
#include<stdio.h>

int w[10],d,n,count,x[10],i;

void sum_of_subsets(int s, int k, int r)
{
        x[k]=1;
        if( s + w[k] == d)
        {
            printf("\n Subset %d = ",++count);
            for(i=0; i<=k; i++)
                if(x[i])
                    printf("%d ",w[i]);
        }
        else if(s+ w[k] + w[k+1] <= d)
            sum_of_subsets(s + w[k], k+1, r-w[k]);

        if ((s + r - w[k] >= d) && ((s + w[k+1]) <= d))
        {
            x[k]=0;
            sum_of_subsets(s, k+1, r-w[k]);
        }
}
```

```
void main()
{
        int sum=0,k;
        printf("Enter the no of elements : ");
        scanf("%d",&n);

        printf("Enter the elements in ascending order : ");
        for(i=0;i<n;i++)
            scanf("%d",&w[i]);
        printf("Enter the sum : ");
        scanf("%d",&d);

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

        if(sum<d||w[0]>d)
            printf("\n No subset possible ");
        else
            sum_of_subsets(0,0,sum);
}
```

Output

```
Enter the no of elements : 6
Enter the elements in ascending order : 1
7
14
22
23
30
Enter the sum : 30
Subset 1 = 1 7 22
Subset 2 = 7 23
Subset 3 = 30

Enter the no of elements : 4
Enter the elements in ascending order:
20
30
40
50
Enter the sum : 15
No subset possible
```