# NRI INSTITUTE OF INFORMATION SCIENCE

# & TECHNOLOGY BHOPAL

## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

# LAB MANUAL

# ANALYSIS AND DESIGN OF ALGORITHM

# (CS-4004)

## BACHELOR OF ENGINEERING (C.B.G.S)

| NIIST BHOPAL | | NRI INSTITUTE OF INFORMATION SCIENCE & TECHNOLOGY<br>DEPT NAME: Computer Science & Engineering | FORM NO | NIIST/A/10 |
|---|---|---|---|---|
| BRANCH | CSE | LIST OF EXPERIMENT | REV. NO | 0 |
| SEMESTER | IV | | REV. DT | 30/06/2011 |
| SUBJECT/CODE :- ANALYSIS AND DESIGN OF ALGORITHM / CS 4004 | | | | |

| S. NO. | LIST OF EXPERIMENT |
|---|---|
| 1 | Write a program for Linear Search |
| 2 | Write a program for Binary Search |
| 3 | Write a program for Matrix Addition |
| 4 | Write a program for Matrix Multiplication |
| 5 | Write a program for Strassen's Matrix Multiplication |
| 6 | Write a program to generate the Fibonacci series using recursion |
| 7 | Write a program for quick sort |
| 8 | Write a program for merge sort |
| 9 | Write a Program for insertion sort |
| 10 | Write a program to implement 0 / 1 Knapsack problem using dynamic programming. |

## Subject Name: ANALYSIS AND DESIGN OF ALGORITHM

## Subject Code: CS-4004

## Course Outcomes:

After successful completion of course, students will be able to:

CO1: Implement various algorithms to solve problems.

CO2: Determine the complexity of implemented algorithms and compare.

CO3: Implement sort and searching algorithms.

CO4: Demonstrate the backtracking problem

CO5: Simulate N-Queen's Problem / 15 Puzzle Problems.

# EXPERIMENT NO. 1

## AIM:

Program for Linear Search

## INTRODUCTION:

In the linear search a key element is searched linearly in the array. A flag is used to keep a track whether the key is found or not. If the key is found the location of the element is displayed.

## SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
void main()
{
      int a[100],i,n,flag=0,item;
      clrscr();
      cout<<"Enter number of elements:";
      cin>>n;
      cout<<”Enter the elements of the array:\n";
      for(i=0;i<=n-1;i++)
      {
            cin<<a[i];
      }
      cout<<”Enter the element to be searched:\n";
      cin>>item;
      for(i=0;i<=n-1;i++)
      {
            if(a[i]==item)
            {
                  cout<<”Element found at location ”<<i+1;
                  flag=1;
            }
      }
      if(flag==0)cout<<”Element not found”;
      getch();
}
```

## SAMPLE OUTPUT:

```
Enter number of elements: 5
Enter the elements:
1
2
3
4
5
Enter number to be searched:5
Element found at position 5
```

# EXPERIMENT NO. 2

**AIM:**

Program for Binary Search

**INTRODUCTION:**

Binary Search is a divide and conquer approach for searching an element in a given sorted array.

A straightforward implementation of binary search is recursive. The initial call uses the indices of the entire array to be searched. The procedure then calculates an index midway between the two indices, determines which of the two subarrays to search, and then does a recursive call to search that subarray. Each of the calls is tail recursive, so a compiler need not make a new stack frame for each call. The variables imin and imax are the lowest and highest inclusive indices that are searched.

```
int binary_search(int A[], int key, int imin, int imax)
{
  // test if array is empty
  if (imax < imin):
    // set is empty, so return value showing not found
    return KEY_NOT_FOUND;
  else
    {
      // calculate midpoint to cut set in half
      int imid = midpoint(imin, imax);

      // three-way comparison
      if (A[imid] > key)
        // key is in lower subset
        return binary_search(A, key, imin, imid-1);
      else if (A[imid] < key)
        // key is in upper subset
        return binary_search(A, key, imid+1, imax);
      else
        // key has been found
        return imid;
    }
}
```

**SOURCE CODE:**

```
#include<iostream.h>

#include<conio.h>

int b_search(int a[],int n, int low, int high)

{

      int mid,t;
```

```
    if(low>high)
          return -1;
    mid=(low+high)/2;
    if(n==a[mid])
    {
          cout<<"Element found at position "<<mid+1;
          return 0;
    }
    if(n<a[mid])
    {
          high=mid-1;
          t=b_search(a,n,low,high);
    }
          if(n>a[mid])
    {
          low=mid+1;
          t=b_search(a,n,low,high);
    }
    return t;
}
void main()
{
int a[50],n,no,result;
clrscr();
cout<<"Enter the number of elements:";
cin>>no;
cout<<"Enter the elements:\n";
for(int i=0;i<=no-1;i++)
    cin>>a[i];
```

```
  cout<<"Enter number to be searched:\n";
  cin>>n;
  result=b_search(a,n,0,no-1);
  if(result==-1)
  cout<<"Element not found";
  getch();
}
```

## SAMPLE OUTPUT:

```
Enter number of elements: 5
Enter the elements:
1
2
3
4
5
Enter number to be searched:5
Element found at position 5
```

# EXPERIMENT NO.3

## AIM:

Program for Matrix Addition

## INTRODUCTION:

The user will be required to input the elements in the 3x3 matrices A and B. The elements of the matrices would be added and stored in matrix C. Nested for loops would be used for entering elements into the matrices and to calculate the sum matrix.

## SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
void main()
{
      int a[3][3],b[3][3],c[3][3],i,j;
      clrscr();
      cout<<"MATRIX A\n";
      //Matrix values to be entered by user
      for(i=0;i<=2;i++)
      {
            for(j=0;j<=2;j++)
            {
                  cout<<"\nEnter element:";
                  cin>>a[i][j];
            }
      }
      cout<<"\nMATRIX B\n";
      for(i=0;i<=2;i++)
      {
            for(j=0;j<=2;j++)
            {
                  cout<<"\nEnter element:";
                  cin>>b[i][j];
            }
      }
      //Addition of two matrices
      cout<<"\nMATRIX C=MATRIX A+MATRIX B\n";
      for(i=0;i<=2;i++)
      {       cout<<"\n\n";
            for(int j=0;j<=2;j++)
            {
                  c[i][j]=a[i][j]+b[i][j];
                  cout<<c[i][j]<<"\t";
            }
      }
      getch();
}
```

## SAMPLE OUTPUT:

MATRIX A:

Enter element:2

Enter element:2

Enter element:2

Enter element:2

Enter element:2

Enter element:2

Enter element:2

Enter element:2

Enter element:2

MATRIX B:

Enter element:2

Enter element:2

Enter element:2

Enter element:2

Enter element:2

Enter element:2

Enter element:2

Enter element:2

Enter element:2

MATRIX C=MATRIX A+MATRIX B

4 4 4

4 4 4

4 4 4

# EXPERIMENT NO.4

## AIM:

Program for matrix multiplication

## INTRODUCTION:

The user will first enter the number of rows and columns of the two matrices. The multiplication of matrices is possible only if the number of columns in the first matrix is equal to the number of rows of the second matrix. If multiplication is possible, user will enter the values in both matrices. The program logic of matrix multiplication is applied to get the resultant matrix.

## SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
void main()
{
	clrscr();
	int i,j,k,m,n,p,q,a[5][5],b[5][5],c[10][10];
	cout<<"\n Enter the number of rows in 1st matrix: ";
	cin>>m;
	cout<<"\n Enter the number of columns in 1st matrix: ";
	cin>>n;
	cout<<"\n Enter the number of rows in 2nd matrix: ";
	cin>>p;
	cout<<"\n Enter the number of columns in 2nd matrix: ";
	cin>>q;
	//Check for possibility of matrix multiplication
	if(n==p)
		cout<<"\n Matrix multiplication possible";
	else
	{
		cout<<"\n Matrix multiplication not possible";
		getch();
		exit(0);
	}
	//Matrix values to be entered by user
	cout<<"\n\n Enter the 1st matrix:\n";
	for(i=0;i<m;i++)
	{
		for(j=0;j<n;j++)
			cin>>a[i][j];
	}
	cout<<"\n\n Enter The 2nd matrix:\n";
	for(i=0;i<p;i++)
	{
		for(j=0;j<q;j++)
			cin>>b[i][j];
	}
	//Display matrices
```

```
	cout<<"\n\n The 1st matrix:\n";
	for(i=0;i<m;i++)
	{
		for(j=0;j<n;j++)
			cout<<a[i][j]<<"\t";

		cout<<"\n";
	}
	cout<<"\n\n The 2nd matrix:\n";
	for(i=0;i<p;i++)
	{
		for(j=0;j<q;j++)
			cout<<b[i][j]<<"\t";

		cout<<"\n";
	}
	//Matrix multiplication
	for(i=0;i<m;i++)
	{
		for(j=0;j<q;j++)
		{
			c[i][j]=0;
			for(k=0;k<n;k++)
				c[i][j]=c[i][j]+(a[i][k]*b[k][j]);
		}
	}
	//Display resultant matrix
	cout<<"\n\n The Resultant matrix:\n";
	for(i=0;i<m;i++)
	{
		for(j=0;j<q;j++)
			cout<<c[i][j]<<"\t";
		cout<<"\n";
	}
	getch();
}
```

## SAMPLE OUTPUT:

Enter the number of rows in 1st matrix:3

Enter the number of columns in 1st matrix:2

Enter the number of rows in 2nd matrix:2

Enter the number of columns in 2nd matrix:3

Matrix multiplication possible

Enter the 1st matrix:

2

3

4

5
6
7
Enter the 2nd matrix:
2
3
4
5
6
7
The 1st matrix:

2 3

4 5

6 7

The 2nd matrix:

2 3 4

5 6 7

The Resultant matrix:

19 24 29

33 42 51

47 60 73

# EXPERIMENT NO. 5

## AIM:

**Program for Strassen's Matrix Multiplication**

## INTRODUCTION:

## Strassen's Matrix Multiplication

**Given t**wo N by N matrices A and B the problem is to compute $C = A \times B$

- **Brute Force** Approach

```
for i:= 1 to N do
  for j:=1 to N do
     C[i,j] := 0;
     for k := 1 to N do
        C[i,j] := C[i,j] + A[i,k] * B[k,j]
```

*Complexity : $O(N^3)$*

- **Divide and Conquer Approach given by Volker Strassen**

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_2 &= (A_{21} + A_{22})B_{11} \\ P_3 &= A_{11}(B_{12} - B_{22}) \\ P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{12})B_{22} \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\ C_{11} &= P_1 + P_4 - P_5 + P_7 \\ C_{12} &= P_3 + P_5 \\ C_{21} &= P_2 + P_4 \\ C_{22} &= P_1 + P_3 - P_2 + P_6 \end{aligned}$$

From

$$T(n) = \begin{cases} 7T(n/2) + cn & \text{if } n > 1 \\ c & \text{if } n = 1 \end{cases}$$

$T(n) = O(n^{\log 7}) = O(n^{2.81})$.

**SOURCE CODE:**

```
#include<iostream.h>
#include<conio.h>
void main(){
clrscr();
  int a[2][2],b[2][2],c[2][2],i,j;
  int m1,m2,m3,m4,m5,m6,m7;

  cout<<"Enter the 4 elements of first matrix: ";
  for(i=0;i<2;i++)
      for(j=0;j<2;j++)
           cin>>a[i][j];

  cout<<"Enter the 4 elements of second matrix: ";
  for(i=0;i<2;i++)
      for(j=0;j<2;j++)
           cin>>b[i][j]);

  cout<<"\nThe first matrix is\n";
  for(i=0;i<2;i++){
      cout<<"\n";
      for(j=0;j<2;j++)
          cout<<"\t"<<a[i][j];
  }

 cout<<"\nThe second matrix is\n";
  for(i=0;i<2;i++){
     cout<<"\n";
      for(j=0;j<2;j++)
          cout<<"\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;

  cout<<"\nAfter multiplication \n";
   for(i=0;i<2;i++){
     cout<<"\n";
      for(j=0;j<2;j++)
        cout<<"\t"<<c[i][j];
   }
```

```
   getch();
}
```

## SAMPLE OUTPUT:

```
Enter the 4 elements of first matrix: 1
2
3
4
Enter the 4 elements of second matrix: 5
6
7
8

The first matrix is

1       2
3       4
The second matrix is

5       6
7       8
After multiplication

19      22
43      50
```

# EXPERIMENT NO.6

## AIM:

Program to generate the Fibonacci series using recursion

## INTRODUCTION:

Fibonacci Series is a series that starts with 0 and 1. Every next number in the series will be the sum of the previous two numbers. Hence, the series would be

0 1 1 2 3 5 8 13 and so on.

In this program the user will give the limit of the series, i.e. the number of terms in the series and the recursive procedure will be executed to generate the series.

## SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
void fseries(int);                                  //Function prototype
void main()
{
      int limit,f0=0,f1=1;
      clrscr();
      cout<<"Enter limit of Fibonacci Series:";
      cin>>limit;
      cout<<"\n\t\t\tFIBONACCI SERIES\n";
      if(limit>2)
      {
            cout<<f0<<"\n"<<f1;
            fseries(limit-2);                       //Calling recursive function
      }
      else if(limit==2)
            cout<<f0<<"\n"<<f1;
      else if(limit==1)
      cout<<f1;
      else
      cout<<"Series not possible";
      getch();
}
void fseries(int p)                                 //Recursive function body
{
      int fib;
      static int f0=0,f1=1;
      if(p==0)
            cout<<"\nEnd of fibonacci series";
      else
      {
            fib=f0+f1;
            f0=f1;
            f1=fib;
            cout<<"\n"<<fib;
            fseries(p-1);
```

```
        }
}
```

## SAMPLE OUTPUT:

Enter limit of Fibonacci Series:10

FIBONACCI SERIES

0

1

1

2

3

5

8

13

21

34

End of fibonacci series

# EXPERIMENT NO.7

## AIM:

Program for insertion sort

## INTRODUCTION:

## Insertion Sort:

Suppose an array A with n elements A[1], A[2]…..A[n] is in memory. The insertion sort algorithm scans A from A[1] to A[n], inserting each element A[k] into its proper position in the previously sorted sub array A[1], A[2]…. A[k-1].

Example:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| List: | 77 | 33 | 44 | 11 | 88 | 22 | 55 |
| Pass I | 77 | 33 | 44 | 11 | 88 | 22 | 55 |
| Pass II | 33 | 77 | 44 | 11 | 88 | 22 | 55 |
| Pass III | 33 | 44 | 77 | 11 | 88 | 22 | 55 |
| Pass IV | 11 | 33 | 44 | 77 | 88 | 22 | 55 |
| Pass V | 11 | 33 | 44 | 77 | 88 | 22 | 55 |
| Pass VI | 11 | 22 | 33 | 44 | 77 | 88 | 55 |
| Pass V | 11 | 22 | 33 | 44 | 55 | 77 | 88 |

## SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[100],n,k,i,j,temp;
clrscr();
printf("How many elements:\n");
scanf("%d",&n);
printf("Enter the elements of array:");
for(i=0;i<=n-1;i++)
{
scanf("%d",&a[i]);
}
for(k=1;k<=n-1;k++)
{
temp=a[k];
j=k-1;
while((temp<a[j])&&(j>=0))
```

```
{
a[j+1]=a[j];
j=j-1;
}
a[j+1]=temp;
}
printf("Elements of array after insertion sort:\n");
for(i=0;i<=n-1;i++)
{
printf("%d\n",a[i]);
}
getch();
}
```

## SAMPLE OUTPUT:

How many elements:

5

Enter the elements of array:78

45

34

12

67

Elements of array after insertion sort:

12

34

45

67

78

# EXPERIMENT NO. 8

## AIM:

**Program for Quick Sort**

## INTRODUCTION:

## Quick Sort:

Choose some key from the list. Call this key the pivot. Then partition the items so that all those with keys less than pivot come in one sub list and all those with greater key come in another. Sort the two reduced list separately and continue the process till the list gets sorted.

Example:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| List: | 24 | 56 | 47 | 35 | 10 | 90 | 82 | 31 |
| Pass I | 10 | 24 | 56 | 47 | 35 | 90 | 82 | 31 |
| Pass II | 10 | 24 | 47 | 35 | 31 | 56 | 82 | 90 |
| Pass III | 10 | 24 | 35 | 31 | 47 | 56 | 82 | 90 |
| Pass IV | 10 | 24 | 31 | 35 | 47 | 56 | 82 | 90 |
| Pass V | 10 | 24 | 31 | 35 | 47 | 56 | 82 | 90 |

## SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#define max 100
int a[max],n,i,l,h;
void main()
{
clrscr();
void input(void);
input();
getch();
}

void input(void)
{
void output(int a[],int n);
void quick_sort(int a[],int l,int h);
cout<<How many elements in the array : ";
cin>>n;
cout<<"\nEnter the elements : \n");
for(i=0;i<=n-1;i++)
```

```
{
cin>>a[i];
}
l=0;
h=n-1;
quick_sort(a,l,h);
cout<<"Sorted Array :\n";
output(a,n);
}

void quick_sort(int a[],int l, int h)
{
      int temp,key,low,high;
      low=l;
      high=h;
      key=a[(low+high)/2];
      do
      {
            while(key>a[low])
            {
                  low++;
            }
            while(key<a[high])
            {
                  high--;
            }
            if(low<=high)
            {
                  temp=a[low];
                  a[low++]=a[high];
                  a[high--]=temp;
            }
      }  while(low<=high);
      if(l<high)
      quick_sort(a,l,high);
      if(low<h)
      quick_sort(a,low,h);
}
void output(int a[],int n)
{
for(i=0;i<=n-1;i++)
{
      cout<<a[i];
}
}
```

## SAMPLE OUTPUT:

How many elements in the array : 5

Enter the elements :

56

34

23

78

45

Sorted Array :

23

34

45

56

78

# EXPERIMENT NO. 9

## AIM:

**Program for Merge Sort**

## INTRODUCTION:

## Merge Sort:

Given a sequence of n elements, A[1],A[2],…….A[n], spit it into two sets A[1],A[2],…….A[n/2] and A[(n/2)+1]……A[n]. Successively select the data element with the smallest key occurring in either of the sets and place this element in a new array.

Example:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| List: | 11 | 20 | 35 | 42 | 9 | 22 | 50 |
| X | 11 | 20 | 35 | 42 | | | |
| Y | 9 | 22 | 50 | | | | |
| Z | 9 | 11 | 20 | 22 | 35 | 42 | 50 |

## SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#define MAX 10

void merge(int *arr, int *temp, int low, int mid, int high);//function
prototype for merge sorting

/*function for merge sorting*/
void m_sort(int *arr, int *temp, int low, int high)
 {
   int mid;
   if(high>low)
    {
      mid = (low+high)/2;//middle element
      m_sort(arr,temp,low,mid);
      m_sort(arr,temp,mid+1,high);
      merge(arr,temp,low,mid+1,high);
    }
 }

void merge(int *arr, int *temp, int low, int mid, int high)
 {
   int i,end,num,pos;
   end = mid-1;
```

```
   pos = low;
   num = high-low+1;
   while((low<=end) && (mid<=high))
    {
      if(arr[low]<=arr[mid])
       {
       temp[pos] = arr[low];
       pos = pos + 1;
       low = low + 1;
       }
      else
       {
       temp[pos] = arr[mid];
       pos = pos + 1;
       mid = mid + 1;
       }
    }
   while(low<=end)
    {
      temp[pos] = arr[low];
      low = low + 1;
      pos = pos + 1;
    }
   while(mid<=high)
    {
      temp[pos] = arr[mid];
      mid = mid + 1;
      pos = pos + 1;
    }
   for(i=0;i<num;i++)
    {
      arr[high] = temp[high];
      high = high - 1;
    }
 }
void main()
 {
   int num,arr[MAX],temp[MAX],i;
   clrscr();
   cout<< "Enter the number of elements :";
   cin>>num;
   if(num>MAX)
    cout<<"\nArray out of bound!";
   else
    { cout<<"\nEnter elements:";
      for(i=0;i<num;i++)
      cin>>arr[i];

      m_sort(arr,temp,0,num);
      cout<<"\nThe list after sorting is :\n";

      for(i=0;i<num;i++)
      cout<<arr[i];
    }
   getch();
 }
```

**SAMPLE OUTPUT:**

Enter the number of elements :5

Enter elements:12

34

55

34

2

The list after sorting is :

2

12

34

34

55

# EXPERIMENT NO.10

## AIM:

**Program to implement 0 / 1 Knapsack problem using dynamic programming.**

## INTRODUCTION:

The **knapsack problem** is a problem in combinatorial optimization: Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

## ALGORITHM :

### Knapsack ( i, j )

//Input: A nonnegative integer i indicating the number of the first

//items being considered and a nonnegative integer j indicating the //Knapsack's capacity.

//Output: The value of an optimal feasible subset of the first I items.

//Note: Uses as global variables input arrays Weights [ 1..n },

//Values [ 1… n ] and table V [ 0…n, 0…W ] whose entries are //initialized with –1's except for row 0 and column 0 initialized with //0's.

Step 1: if V [i, j ] < 0 {

Step 2: if j < Weights[i] then value = Knapsack ( i-1, j )

Step 3: else {

Step 4: value = max ( Knapsack ( i-1, j )

Step 5: Values[I] + Knapsack ( i1, j-Weights[i] ) )

} //end else

Step 6: V[i j] = value

} //end if

Step 7: return V [ i, j ]

## SOURCE CODE:

```
# include <stdio.h>
# include <conio.h>
    int m;
    int n;
    float w[30];
    float p[30];
    float x[30];
    float totprof ;

float DKnapSack(int k, int m)
{
```

```
    float res, res1, res2;
    if(k==n)
       res = (w[k] <= m) ? p[k] : 0 ;
    else if(w[k] > m)
       res = DKnapSack(k+1, m);
    else
    {
       res1 = p[k] + DKnapSack(k+1, m-w[k]);
       res2 = DKnapSack(k+1, m);
       res = (res1 > res2) ? res1 : res2 ;
    }
    return(res);
}

void main()
{
    int k;
    float p1,p2, wt;

    clrscr();
    printf("\n Enter the Knapsack Capacity : ");
    scanf("%d", &m);
    printf("\n Enter the number of Objects : ");
    scanf("%d", &n);
    for(k=1; k<=n; k++)
    {
       printf("\n Enter the Weight & Profit for Object %d : ", k);
       scanf("%f%f", &w[k], &p[k]);
    }

    totprof = DKnapSack(1, m);
    wt=0;
    for(k=1; k<=n-1; k++)
    {
       p1=DKnapSack(k, m-wt);
       p2=DKnapSack(k+1, m-wt);
       if(p1==p2)
          x[k] = 0;
       else
       {  x[k] = 1; wt=wt+w[k];  }
    }
    p1=DKnapSack(n, m-wt);
    x[n] = (p1==0) ? 0 : 1 ;

    printf("\n Maximum Profit : %f", totprof);
    printf("\n Solution Vector : \n");
    for(k=1; k<=n; k++)
       printf("%6.2f",x[k]);

    getch();
}
```

## SAMPLE OUTPUT:

Enter the number of objects : 3

Enter the weights :

1 2 2

Enter the profits :

18 16 6

Enter the Knapsack capacity: 4

Optimal solution = 34