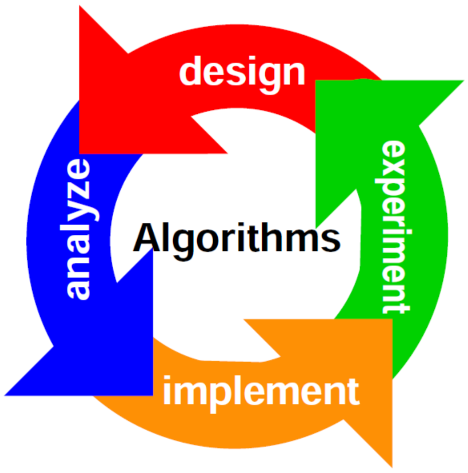


LAB MANUAL

of

ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY

(CSL52)

for

V Semester,

Department of CSE

Siddaganga Institute of Technology

Tumkur 572103

Prabodh C P
Asst. Professor,
Dept of CSE,
SIT, Tumkur 572103

Question 1a

Use divide and conquer method to recursively implement Binary Search

```
/******************************************************************************
*File       : BinarySearch.c
*Description      : Program to perform Binary Search using Divide & Conquer
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : Friday 28 November 2013
******************************************************************************/

#include<stdio.h>
int fnBinSearch(int A[], int k, int iLow,int iHigh);

/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/

int main(void)
{
      int iaArr[20],iNum,iKey;
      int i,iPos=0;
      printf("\nEnter the size of the array\n");
      scanf("%d",&iNum);
      printf("\nEnter the elements of the array in ascending order:\n");
      for(i=0;i<iNum;i++)
            scanf("%d",&iaArr[i]);
      printf("\nenter the key element\n");
      scanf("%d",&iKey);

      iPos=fnBinSearch(iaArr,iKey,0,iNum-1);

      if(iPos==-1)
            printf("\nElement not found\n");
      else
            printf("\nElement found at position %d\n",iPos+1);

      return 0;
}

/******************************************************************************
*Function   : fnBinSearch
*Description      : Function to perform Binary Search using Divide & Conquer
*Input parameters:
*     int A[]     - array of elements in ascending order
*     int k - key element to be searched
*     int iLow - lower bound
*     int iHigh - upper bound
*RETURNS    : position of the element if found or -1 otherwise
******************************************************************************/

int fnBinSearch(int A[], int k, int iLow,int iHigh)
{
      int iMid;
      if(iLow<=iHigh)
      {
            iMid=(iLow+iHigh)/2;
            if(k==A[iMid])
```

```
                return iMid;
            if(k<A[iMid])
                return fnBinSearch(A,k,iLow,iMid-1);
            if(k>A[iMid])
                return fnBinSearch(A,k,iMid+1,iHigh);
        }
        else
            return -1;
}


/******************************************************************************
OUTPUT

SAMPLE 1

Enter the size of the array
5

Enter the elements of the array in ascending order:
1 3 5 7 9

enter the key element
7

Element found at position 4

SAMPLE 2

Enter the size of the array
4

Enter the elements of the array in ascending order:
1 2 3 4

enter the key element
5

Element not found

******************************************************************************/
```

Question 1a

Use divide and conquer method to recursively implement Linear Search

```
/******************************************************************************
*File       : LinearSearch.c
*Description      : Program to perform Linear Search
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : Friday 13 December 2013
******************************************************************************/
#include<stdio.h>
int fnLinSearch(int [], int, int);

/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{
      int iaArr[20],iNum,iKey;
      int i,iPos=0;
      printf("\nEnter the size of the array\n");
      scanf("%d",&iNum);
      printf("\nEnter the elements of the array:\n");
      for(i=0;i<iNum;i++)
            scanf("%d",&iaArr[i]);
      printf("\nenter the key element\n");
      scanf("%d",&iKey);

      iPos=fnLinSearch(iaArr,iKey,iNum);

      if(iPos==-1)
            printf("\nElement not found\n");
      else
            printf("\nElement found at position %d\n",iPos);

      return 0;
}

/******************************************************************************
*Function   : fnLinSearch
*Description      : Function to perform Linear Search
*Input parameters:
*     int A[]     - array of elements in ascending order
*     int k - key element to be searched
*     int iN - no of elements to be searched
*RETURNS    : position of the element if found or -1 otherwise
******************************************************************************/

int fnLinSearch(int A[], int k, int iN)
{
      if( iN ==0)
            return -1;
      else if( k == A[iN-1])
            return iN;
      else
            return fnLinSearch(A,k,iN-1);
}
```

Question 1b

Use divide and conquer method to recursively implement and to find the maximum and minimum in a given list of n elements.

```
/******************************************************************************
*File       : MaxMin.c
*Description      : Program to find the maximum and minimum in a given list
*                           of n elements using divide and conquer.
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : 11 Nov 2013
******************************************************************************/

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

void fnRecMaxMin(int [], int , int, int*, int*);
/******************************************************************************
*Function   : main
*Input parameters:
*     int argc - no of commamd line arguments
*     char **argv - vector to store command line argumennts
*RETURNS    :     0 on success
******************************************************************************/
int main( int argc, char **argv)
{
      int iaArr[500000],iNum,i;
      int iMax=0,iMin=0;
      printf("\nEnter the size of the array\n");
      scanf("%d",&iNum);
      printf("\nEnter the elements of the array:\n");
      for(i=0;i<iNum;i++)
            scanf("%d",&iaArr[i]);

      fnRecMaxMin(iaArr, 0, iNum-1, &iMax, &iMin);

      printf("\nMax Element = %d\nMin Element = %d\n", iMax, iMin);
      return 0;
}

/******************************************************************************
*Function   : fnRecMaxMin
*Description      : Function to find maximum and minimum elements in an
*                           array using Divide and Conquer,
*Input parameters:
*     int a[] - iaArray to hold integers
*     int low     - start index of the array to be sorted
*     int high- end index of the array to be sorted
*     int *max - pointer to hold the max element
*     int *min - pointer to hold the min element
*RETURNS    : no value
******************************************************************************/

void fnRecMaxMin(int a[],int low,int high, int *max, int *min)
{
      int mid,max1,max2,min1,min2;
      if(high-low == 1)
      {
            if(a[low] > a[high])
            {
```

```
            *max = a[low];
            *min = a[high];
        }
        else
        {
            *max = a[high];
            *min = a[low];
        }
    }
    else if(low == high)
    {
        *min = *max = a[low];
    }
    else if(low<high)
    {
        mid=(low+high)/2;
        fnRecMaxMin(a,low,mid,&max1,&min1);
        fnRecMaxMin(a,mid+1,high,&max2,&min2);
        if(max1 > max2)
            *max = max1;
        else
            *max = max2;

        if(min1 < min2)
            *min = min1;
        else
            *min = min2;
    }
}
/*******************************************************************************
OUTPUT

SAMPLE 1
Enter the size of the array
6

Enter the elements of the array:
7 2 3 6 5 4

Max Element = 7
Min Element = 2

SAMPLE 2
Enter the size of the array
7

Enter the elements of the array:
7 1 2 6 5 3 4

Max Element = 7
Min Element = 1

SAMPLE 3
Enter the size of the array
7
Enter the elements of the array:
1 3 5 7 2 4 6

Max Element = 7
Min Element = 1
*******************************************************************************/
```

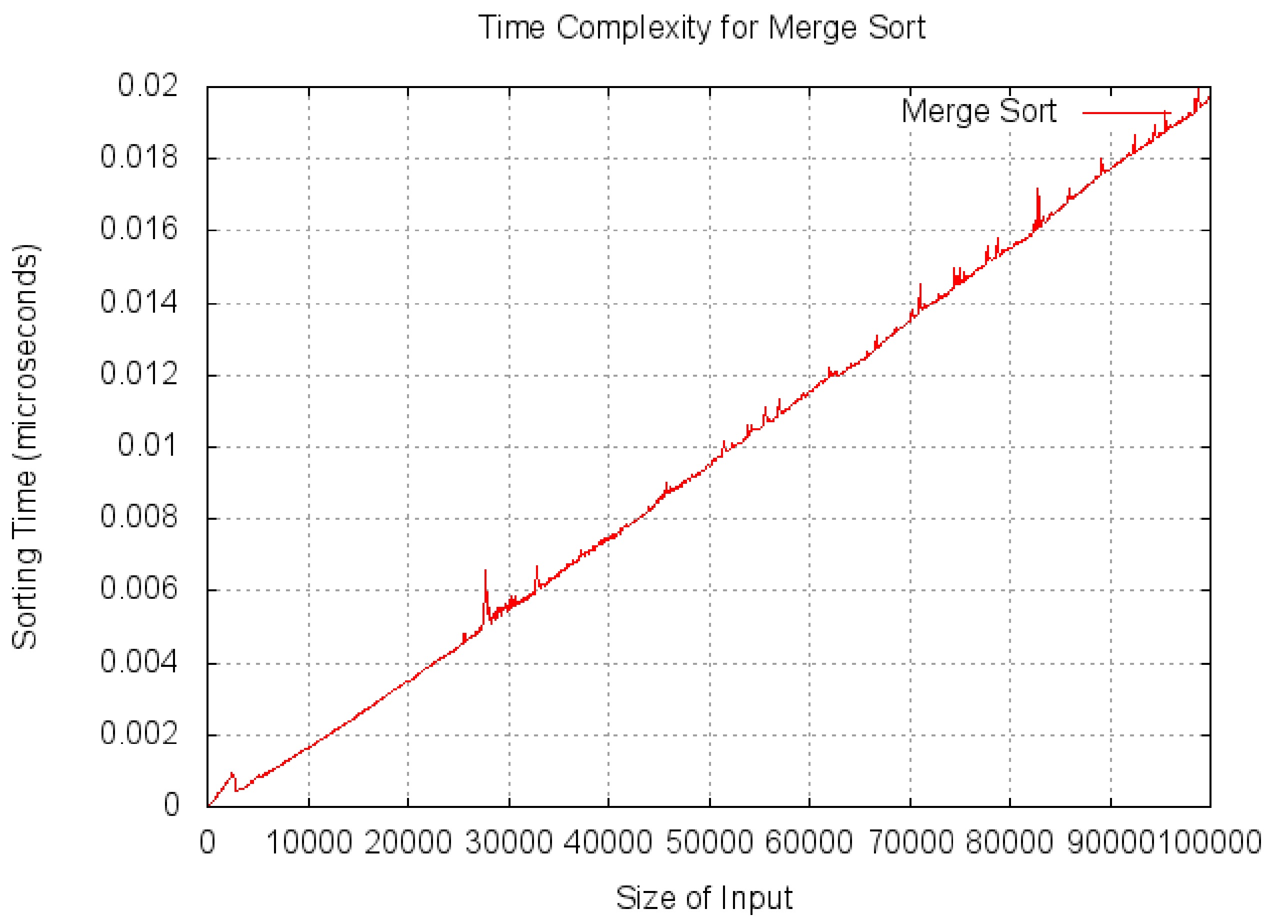
Question 2

Sort a given set of elements using the Merge sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and 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.

```
/******************************************************************************
*File       : MergeSort.c
*Description      : Program to sort an array using Merge Sort
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : 11 Nov 2013
******************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>


void fnGenRandInput(int [], int);
void fnDispArray( int [], int);
void fnMerge(int [], int ,int ,int);
void fnMergeSort(int [], int , int);

/******************************************************************************
*Function   : main
*Input parameters:
*     int argc - no of commamd line arguments
*     char **argv - vector to store command line argumennts
*RETURNS    :     0 on success
******************************************************************************/

int main( int argc, char **argv)
{

      FILE *fp;
      struct timeval tv;
      double dStart,dEnd;
      int iaArr[500000],iNum,iPos,iKey,i,iChoice;

    for(;;)
    {
        printf("\n1.Plot the Graph\n2.MergeSort\n3.Exit");
        printf("\nEnter your choice\n");
        scanf("%d",&iChoice);

        switch(iChoice)
        {
            case 1:
                fp = fopen("MergePlot.dat","w");

                for(i=100;i<100000;i+=100)
                {
                    fnGenRandInput(iaArr,i);

                    gettimeofday(&tv,NULL);
                    dStart = tv.tv_sec + (tv.tv_usec/1000000.0);
                    fnMergeSort(iaArr,0,i-1);

                    gettimeofday(&tv,NULL);
                    dEnd = tv.tv_sec + (tv.tv_usec/1000000.0);

                    fprintf(fp,"%d\t%lf\n",i,dEnd-dStart);

                }
                fclose(fp);

                printf("\nData File generated and stored in file <
MergePlot.dat >.\n Use a plotting utility\n");
            break;

            case 2:
                printf("\nEnter the number of elements to sort\n");
                scanf("%d",&iNum);
                printf("\nUnsorted Array\n");
                fnGenRandInput(iaArr,iNum);
                fnDispArray(iaArr,iNum);
                fnMergeSort(iaArr,0,iNum-1);
                printf("\nSorted Array\n");
                fnDispArray(iaArr,iNum);
            break;

            case 3:
                exit(0);
        }
    }

      return 0;
}

/*******************************************************************************
*Function   : fnMerge
*Description      : Function to merge two sorted arrays
*Input parameters:
*     int a[] - iaArray to hold integers
*     int low     - start index of the subiaArray to be sorted
*     int mid     - mid index of the subiaArray to be sorted
*     int right   - end index of the subiaArray to be sorted
*RETURNS    : no value
*******************************************************************************/

void fnMerge(int a[], int low,int mid,int high)
{
      int  i,k,j,b[500000];
      i=k=low;
      j=mid+1;
      while(i<=mid && j<=high)
      {
            if(a[i]<a[j])
                  b[k++]=a[i++];
            else
                  b[k++]=a[j++];
      }
      while(i<=mid)
            b[k++]=a[i++];
      while(j<=high)
            b[k++]=a[j++];
```

```
	for(i=low;i<k;i++)
	a[i]=b[i];
}

/******************************************************************************
*Function	: fnMergeSort
*Description	: Function to sort elements in an iaArray using Quick Sort
*Input parameters:
*	int a[] - iaArray to hold integers
*	int low	- start index of the array to be sorted
*	int high- end index of the array to be sorted
*RETURNS	: no value
******************************************************************************/

void fnMergeSort(int a[],int low,int high)
{
	int mid;
	if(low<high)
	{
		mid=(low+high)/2;
		fnMergeSort(a,low,mid);
		fnMergeSort(a,mid+1,high);
		fnMerge(a,low,mid,high);
	}
}

/******************************************************************************
*Function	: GenRandInput
*Description	: Function to generate a fixed number of random elements
*Input parameters:
*	int X[] - array to hold integers
*	int n - no of elements in the array
*RETURNS	:no return value
******************************************************************************/

void fnGenRandInput(int X[], int n)
{
	int i;

	srand(time(NULL));
	for(i=0;i<n;i++)
	{
		X[i] = rand()%10000;
	}
}
/******************************************************************************
*Function	: DispArray
*Description	: Function to display elements of an array
*Input parameters:
*	int X[] - array to hold integers
*	int n - no of elements in the array
*RETURNS	: no return value
******************************************************************************/

void fnDispArray( int X[], int n)
{
	int i;

	for(i=0;i<n;i++)
		printf(" %5d \n",X[i]);
}
```

```
/*****************************************************************************
gnuplot> plot "./MergePlot.txt"
gnuplot> plot "./MergePlot.txt" with lines
gnuplot> plot "./MergePlot.txt" with linespoints
*****************************************************************************/
MergePlot.gpl

# Gnuplot script file for plotting data in file "MergePlot.dat"
# This file is called       MergePlot.gpl
set terminal png font arial
set title "Time Complexity for Merge Sort"
set autoscale
set xlabel "Size of Input"
set ylabel "Sorting Time (microseconds)"
set grid
set output "MergePlot.png"
plot "MergePlot.dat" t 'Merge Sort' with lines

Execution
$gnuplot MergePlot.gpl
```

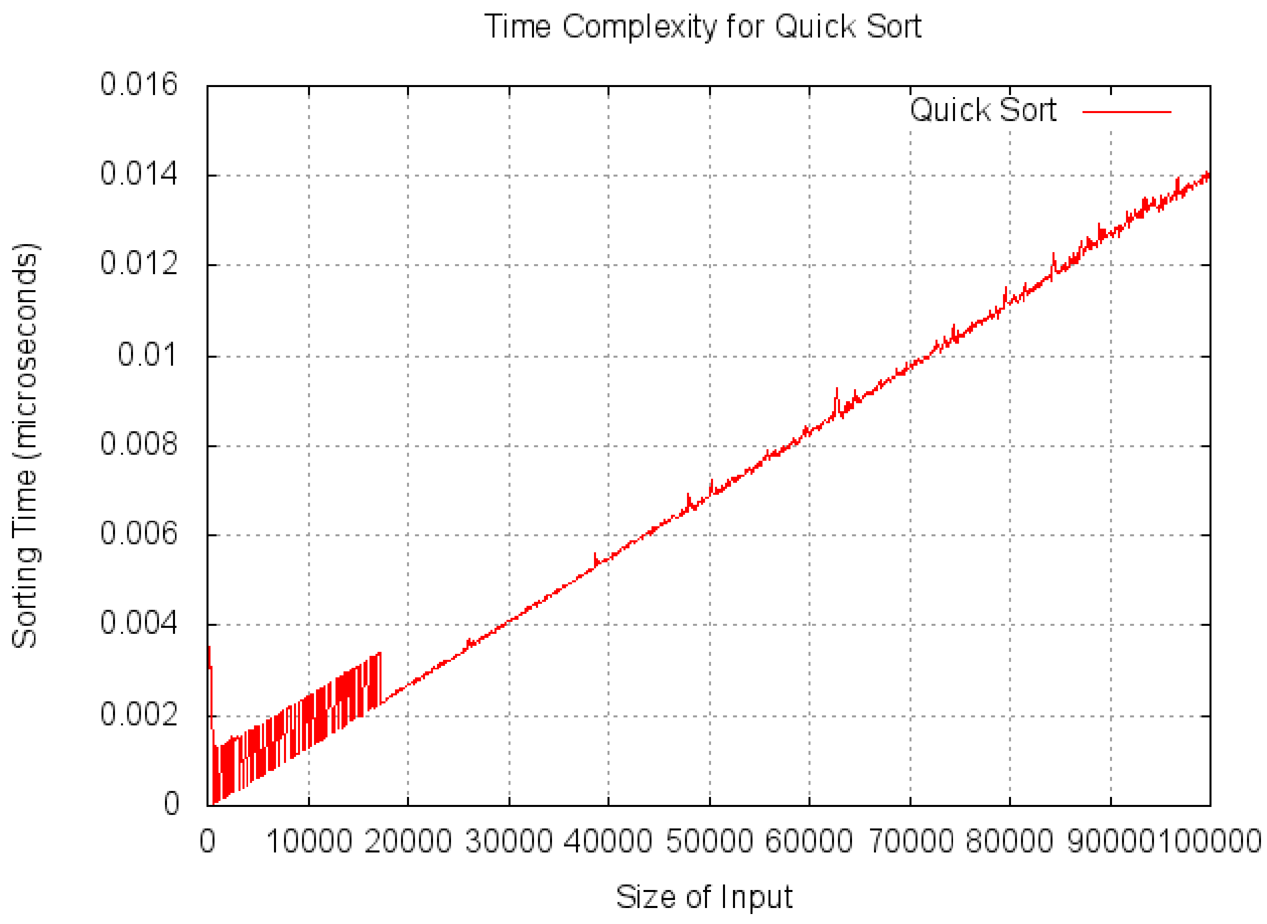


Question 3

Sort a given set of elements using the Quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and 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.

```
/******************************************************************************
*File       : QuickSort.c
*Description      : Program to sort an array using Quick Sort
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : 11 Nov 2013
******************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

void fnGenRandInput(int [], int);
void fnDispArray( int [], int);
int fnPartition(int [], int , int );
void fnQuickSort(int [], int , int );
inline void fnSwap(int*, int*);


inline void fnSwap(int *a, int *b)
{
      int t = *a; *a = *b; *b = t;
}

/******************************************************************************
*Function   : main
*Input parameters:
*     int argc - no of commamd line arguments
*     char **argv - vector to store command line argumennts
*RETURNS    :     0 on success
******************************************************************************/
int main( int argc, char **argv)
{

      FILE *fp;
      struct timeval tv;
      double dStart,dEnd;
      int iaArr[500000],iNum,iPos,iKey,i,iChoice;

    for(;;)
    {
        printf("\n1.Plot the Graph\n2.QuickSort\n3.Exit");
        printf("\nEnter your choice\n");
        scanf("%d",&iChoice);
        switch(iChoice)
        {
            case 1:
                fp = fopen("QuickPlot.dat","w");

                for(i=100;i<100000;i+=100)
                {
                    fnGenRandInput(iaArr,i);

                    gettimeofday(&tv,NULL);
                    dStart = tv.tv_sec + (tv.tv_usec/1000000.0);

                    fnQuickSort(iaArr,0,i-1);

                    gettimeofday(&tv,NULL);
                    dEnd = tv.tv_sec + (tv.tv_usec/1000000.0);

                    fprintf(fp,"%d\t%lf\n",i,dEnd-dStart);
                }
                fclose(fp);

                printf("\nData File generated and stored in file <
QuickPlot.dat >.\n Use a plotting utility\n");
            break;

            case 2:
                printf("\nEnter the number of elements to sort\n");
                scanf("%d",&iNum);
                printf("\nUnsorted Array\n");
                fnGenRandInput(iaArr,iNum);
                fnDispArray(iaArr,iNum);
                fnQuickSort(iaArr,0,iNum-1);
                printf("\nSorted Array\n");
                fnDispArray(iaArr,iNum);
            break;

            case 3:
                exit(0);
        }
    }

      return 0;
}

/******************************************************************************
*Function   : fnPartition
*Description      : Function to partition an iaArray using First element as
Pivot
*Input parameters:
*     int a[] - iaArray to hold integers
*     int l - start index of the subiaArray to be sorted
*     int r - end index of the subiaArray to be sorted
*RETURNS    : integer value specifying the location of partition
******************************************************************************/
int fnPartition(int a[], int l, int r)
{
      int i,j,temp;
      int p;

      p = a[l];
      i = l;
      j = r+1;

      do
      {
            do { i++; } while (a[i] < p);
            do { j--; } while (a[j] > p);
            fnSwap(&a[i], &a[j]);
      }
      while (i<j);

      fnSwap(&a[i], &a[j]);
      fnSwap(&a[l], &a[j]);

      return j;
}

/********************************************************************************
*Function   : fnQuickSort
*Description      : Function to sort elements in an iaArray using Quick Sort
*Input parameters:
*     int a[] - iaArray to hold integers
*     int l - start index of the subiaArray to be sorted
*     int r - end index of the subiaArray to be sorted
*RETURNS    : no value
********************************************************************************/
void fnQuickSort(int a[], int l, int r)
{
      int s;
      if (l < r)
      {
            s = fnPartition(a, l, r);
            fnQuickSort(a, l, s-1);
            fnQuickSort(a, s+1, r);
      }
}

/********************************************************************************
*Function   : GenRandInput
*Description      : Function to generate a fixed number of random elements
*Input parameters:
*     int X[] - array to hold integers
*     int n - no of elements in the array
*RETURNS    :no return value
********************************************************************************/
void fnGenRandInput(int X[], int n)
{
      int i;
      srand(time(NULL));
      for(i=0;i<n;i++)
      {
            X[i] = rand()%10000;
      }
}
/********************************************************************************
*Function   : DispArray
*Description      : Function to display elements of an array
*Input parameters:
*     int X[] - array to hold integers
*     int n - no of elements in the array
*RETURNS    : no return value
********************************************************************************/
void fnDispArray( int X[], int n)
{
      int i;
      for(i=0;i<n;i++)
            printf(" %5d \n",X[i]);
}
```

```
/*******************************************************************************
gnuplot> plot "./QuickPlot.txt"
gnuplot> plot "./QuickPlot.txt" with lines
******************************************************************************/

QuickPlot.gpl

# Gnuplot script file for plotting data in file "QuickPlot.dat"
# This file is called       QuickPlot.gpl
set terminal png font arial
set title "Time Complexity for Quick Sort"
set autoscale
set xlabel "Size of Input"
set ylabel "Sorting Time (microseconds)"
set grid
set output "QuickPlot.png"
plot "QuickPlot.dat" t 'Quick Sort' with lines

Execution
$gnuplot QuickPlot.gpl
******************************************************************************/
```

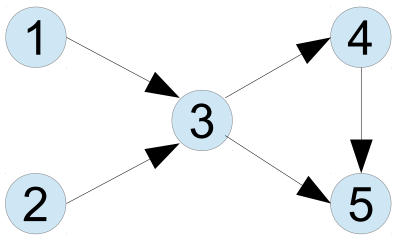


Question 4a

Obtain the Topological ordering of vertices in a given digraph.

```
/******************************************************************************
*File       : TopologicalSorting.c
*Description      : Program to find Topological ordering of nodes in a DAG
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : 11 Nov 2013
******************************************************************************/
#include <stdio.h>

const int MAX = 10;
void fnTopological(int a[MAX][MAX], int n);

/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{
      int a[MAX][MAX],n;
      int i,j;

      printf("Topological Sorting Algorithm -\n");
      printf("\nEnter 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",&a[i][j]);

      fnTopological(a,n);
      printf("\n");
      return 0;
}
/******************************************************************************
*Function   : fnTopological
*Description      : Function to find Topological ordering of nodes in a DAG
*Input parameters:
*     int a[MAX][MAX] - adjacency matrix of the input graph
*     int n - no of vertices in the graph
*RETURNS    : no value
******************************************************************************/

void fnTopological(int a[MAX][MAX], int n)
{
      int in[MAX], out[MAX], stack[MAX], top=-1;
      int i,j,k=0;

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

```
    while(1)
    {
        for (i=0;i<n;i++)
        {
            if (in[i] == 0)
            {
                stack[++top] = i;
                in[i] = -1;
            }
        }

        if (top == -1)
            break;

        out[k] = stack[top--];

        for (i=0;i<n;i++)
        {
            if (a[out[k]][i] == 1)
                in[i]--;
        }
        k++;
    }

    printf("Topological Sorting (JOB SEQUENCE) is:- \n");
    for (i=0;i<k;i++)
        printf("%d ",out[i] + 1);
}

/******************************************************************************
```



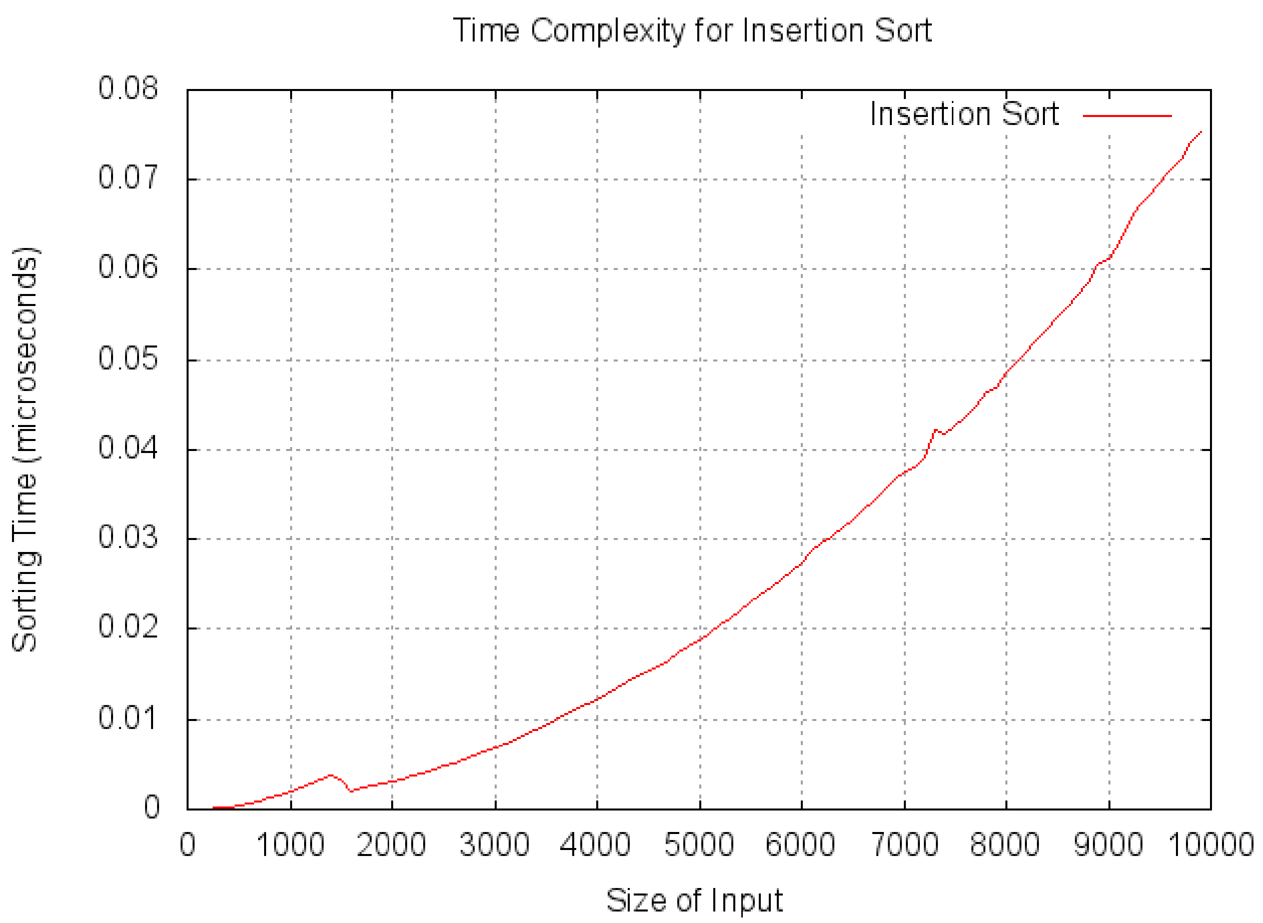
```
Input Graph : 5 vertices
    0 0 1 0 0
    0 0 1 0 0
    0 0 0 1 1
    0 0 0 0 1
    0 0 0 0 0

Topological Sorting (JOB SEQUENCE) is:-
2 1 3 4 5

******************************************************************************/
```

Question 4b

Sort a given set of elements using Insertion sort method.

```
/******************************************************************************
*File       : InsertionSort.c
*Description      : Program to sort an array using Insertion Sort
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : 22 Nov 2013
******************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

void fnGenRandInput(int [], int);
void fnDispArray( int [], int);
void fnInsertionSort(int [], int);

/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{

      FILE *fp;
      struct timeval tv;
      double dStart,dEnd;
      int iaArr[100000],iNum,iPos,iKey,i,iChoice;

    for(;;)
    {
        printf("\n1.Plot the Graph\n2.InsertionSort\n3.Exit");
        printf("\nEnter your choice\n");
        scanf("%d",&iChoice);

        switch(iChoice)
        {
            case 1:
                fp = fopen("InsertionPlot.dat","w");

                for(i=100;i<10000;i+=100)
                {
                    fnGenRandInput(iaArr,i);

                    gettimeofday(&tv,NULL);
                    dStart = tv.tv_sec + (tv.tv_usec/1000000.0);

                        fnInsertionSort(iaArr,i);

                    gettimeofday(&tv,NULL);
                    dEnd = tv.tv_sec + (tv.tv_usec/1000000.0);

                    fprintf(fp,"%d\t%lf\n",i,dEnd-dStart);

                }
                fclose(fp);
                printf("\nData File generated and stored in file <
InsertionPlot.dat >.\n Use a plotting utility\n");
            break;

            case 2:
                printf("\nEnter the number of elements to sort\n");
                scanf("%d",&iNum);
                printf("\nUnsorted Array\n");
                fnGenRandInput(iaArr,iNum);
                fnDispArray(iaArr,iNum);
                  fnInsertionSort(iaArr,iNum);
                printf("\nSorted Array\n");
                fnDispArray(iaArr,iNum);
            break;

            case 3:
                exit(0);
        }
    }
    return 0;
}

/******************************************************************************
*Function   : fnInsertionSort
*Description      : Function to sort an array using Insertion Sort
*Input parameters:
*     int A[] - iaArray to hold integers
*     int n - no of elements in the array
*RETURNS    : no value
******************************************************************************/

void fnInsertionSort(int A[], int n)
{
    int i, j, iKey;
    for(i=1;i<n;i++)
    {
        iKey = A[i];
        j = i-1;
        while(j>=0 && A[j] > iKey)
        {
              A[j+1] = A[j];
              j--;
        }
        A[j+1] = iKey;
    }
}

/******************************************************************************
*Function   : GenRandInput
*Description      : Function to generate a fixed number of random elements
*Input parameters:
*     int X[] - array to hold integers
*     int n - no of elements in the array
*RETURNS    :no return value
******************************************************************************/

void fnGenRandInput(int X[], int n)
{
      int i;
      srand(time(NULL));
      for(i=0;i<n;i++)
```

```
        {
                X[i] = rand()%10000;
        }
}
/********************************************************************************
*Function   : DispArray
*Description      : Function to display elements of an array
*Input parameters:
*      int X[] - array to hold integers
*      int n - no of elements in the array
*RETURNS     : no return value
********************************************************************************/
void fnDispArray( int X[], int n)
{
        int i;
        for(i=0;i<n;i++)
                printf(" %5d \n",X[i]);
}
/******************************************************************************
InsertionPlot.gpl

# Gnuplot script file for plotting data in file "InsertionPlot.dat"
# This file is called       InsertionPlot.gpl
set terminal png font arial
set title "Time Complexity for Insertion Sort"
set autoscale
set xlabel "Size of Input"
set ylabel "Sorting Time (microseconds)"
set grid
set output "InsertionPlot.png"
plot "InsertionPlot.dat" t 'Insertion Sort' with lines

Execution
$gnuplot InsertionPlot.gpl
********************************************************************************/
```



Question 5a

Print all the nodes reachable from a given starting node in a given digraph using Depth First Search method.

```
/******************************************************************************
*File       : DFS.cpp
*Description: Program to find all nodes reachable from a given node using DFS
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : Wednesday 27 November 2013
******************************************************************************/
#include <iostream>
#include <cstdlib>
using namespace std;

const int MAX = 100;

void fnDepthFirstSearch(int currentVertex, int v[MAX], int g[MAX][MAX], int n);

/******************************************************************************
*Function    : main
*Input parameters: no parameters
*RETURNS     :     0 on success
******************************************************************************/
int main(void)
{
      int i,j,k;
      int visited[MAX];
      int graph[MAX][MAX];
      int numVert, Vert;

      cout << "Enter the number of vertices : ";
      cin >> numVert;

      for (i=0; i<numVert; i++)
            visited[i] = 0;

      cout << "Enter the adjacency matrix :\n";

      for (i=0; i<numVert; i++)
            for (j=0; j<numVert; j++)
                  cin >> graph[i][j];

      cout << "Enter the source vertex : ";
      cin >> Vert;

    fnDepthFirstSearch(Vert,visited,graph,numVert);

    for (k=0; k<numVert; k++)
    {
        if(visited[k])
        {
            cout << "\nVertex " << k+1 << " reachable " << endl;
        }
        else
        {
            cout << "\nVertex " << k+1 << " not reachable " << endl;
        }
    }
    return 0;
}

/******************************************************************************
*Function   : fnDepthFirstSearch
*Description      : Function to perform DFS traversal and mark visited vertices
*Input parameters:
*     int currentVertex - source vertex
*     int v[]           - vector to store visited information
*     int g[][]   - adjacency matrix of the graph
*     int n       - no of vertices
*RETURNS    : void
******************************************************************************/

void fnDepthFirstSearch(int currentVertex, int v[MAX], int g[MAX][MAX], int n)
{

      int i;

      v[currentVertex] = 1;

      for (i=0; i<n; i++)
            if (g[currentVertex][i] && !v[i])
                  fnDepthFirstSearch(i,v,g,n);
}

/******************************************************************************
OUTPUT
SAMPLE 1
Enter the number of vertices : 4
Enter the adjacency matrix :
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Enter the starting vertex : 1

Vertex 1 reachable
Vertex 2 reachable
Vertex 3 reachable
Vertex 4 reachable

SAMPLE 2

Enter the number of vertices : 4
Enter the adjacency matrix :
0 1 0 0
1 0 0 0
0 0 0 1
0 0 1 0
Enter the starting vertex : 1

Vertex 1 reachable
Vertex 2 reachable
Vertex 3 not reachable
Vertex 4 not reachable

SAMPLE 3

Enter the number of vertices : 4
Enter the adjacency matrix :
```

```
0 1 0 0
0 0 1 0
0 0 0 1
0 0 0 0
Enter the source vertex : 2

Vertex 1 not reachable
Vertex 2 not reachable
Vertex 3 reachable
Vertex 4 reachable

SAMPLE 4

Enter the number of vertices : 4
Enter the adjacency matrix :
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
Enter the source vertex : 2

Vertex 1 reachable
Vertex 2 reachable
Vertex 3 reachable
Vertex 4 reachable
******************************************************************************/
```

Question 5b

Print all the nodes reachable from a given starting node in a given digraph using Breadth First Search method.

```
/******************************************************************************
*File       : BFS.cpp
*Description: Program to find all nodes reachable from a given node using BFS
*Author          : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : Wednesday 27 November 2013
******************************************************************************/
#include <iostream>
#include <cstdlib>
using namespace std;

const int MAX = 100;
void fnBreadthFirstSearchReach(int vertex, int g[MAX][MAX], int v[MAX], int n);

class Queue
{
      private:
            int cQueue[MAX];
            int front, rear;

      public:
            Queue();
            ~Queue();
            int enqueue(int data);
            int dequeue();
            int empty() { return front == -1 ? 1 : 0;  };
};

/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{
      int i,j;
      int graph[MAX][MAX];
      int visited[MAX];
      int numVert;
      int startVert;


      cout << "Enter the number of vertices : ";
      cin >> numVert;

      cout << "Enter the adjacency matrix :\n";
      for (i=0; i<numVert; i++)
            visited[i] = 0;

      for (i=0; i<numVert; i++)
            for (j=0; j<numVert; j++)
                  cin >> graph[i][j];

      cout << "Enter the starting vertex : ";
      cin >> startVert;
      fnBreadthFirstSearchReach(startVert-1,graph,visited,numVert);

      cout << "Vertices which can be reached from vertex " << startVert << " 
are :-" << endl;
      for (i=0; i<numVert; i++)
            if (visited[i])
                  cout << i+1 << ", ";
      cout << endl;
      return 0;
}
/*Constructor*/
Queue::Queue()
{
      front = rear = -1;
}

/*Destructor*/
Queue::~Queue()
{
}

/********************************************************************************
*Function   : enqueue
*Description      : Function to insert an element at the rear of a Queue
*Input parameters:
*int data   - element to be inserted into the queue
*RETURNS    : returns 1 on success and 0 if queue is full
********************************************************************************/

int Queue::enqueue(int data)
{
      if (front == (rear+1)%MAX)
            return 0;

      if (rear == -1)
            front = rear = 0;
      else
            rear = (rear+1)%MAX;

      cQueue[rear] = data;
      return 1;
}

/********************************************************************************
*Function   : dequeue
*Description      : Function to delete an element from the front of a Queue
*Input parameters: no parameters
*RETURNS    : returns element deleted on success and -1 if queue is empty
********************************************************************************/

int Queue::dequeue()
{
      int data;

      if (front == -1)
            return -1;

      data = cQueue[front];

      if (front == rear)
            front = rear = -1;
      else
            front = (front+1)%MAX;

      return data;
}

/******************************************************************************
*Function   : fnBreadthFirstSearchReach
*Description      : Function to perform BFS traversal and mark visited vertices
*Input parameters:
*     int vertex  - source vertex
*     int g[][]   - adjacency matrix of the graph
*     int v[]           - vector to store visited information
*     int n       - no of vertices
*RETURNS    : void
******************************************************************************/

void fnBreadthFirstSearchReach(int vertex, int g[MAX][MAX], int v[MAX], int n)
{
      Queue verticesVisited;
      int frontVertex;
      int i;

      v[vertex] = 1;
      verticesVisited.enqueue(vertex);

      while (!verticesVisited.empty())
      {
            frontVertex = verticesVisited.dequeue();
            for (i=0; i<n; i++)
            {
                  if (g[frontVertex][i] && !v[i])
                  {
                        v[i] = 1;
                        verticesVisited.enqueue(i);
                  }
            }
      }
}


/******************************************************************************
OUTPUT
SAMPLE 1

Enter the number of vertices : 4
Enter the adjacency matrix :
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Enter the starting vertex : 1
Vertices which can be reached from vertex 1 are :-
1, 2, 3, 4,

SAMPLE 2

Enter the number of vertices : 4
Enter the adjacency matrix :
0 1 0 0
1 0 0 0
```

```
0 0 0 1
0 0 1 0
Enter the starting vertex : 1
Vertices which can be reached from vertex 1 are :-
1, 2,

SAMPLE 3

Enter the number of vertices : 4
Enter the adjacency matrix :
0 1 0 0
0 0 1 0
0 0 0 1
0 0 0 0
Enter the starting vertex : 2
Vertices which can be reached from vertex 2 are :-
2, 3, 4,

SAMPLE 4

Enter the number of vertices : 4
Enter the adjacency matrix :
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
Enter the starting vertex : 2
Vertices which can be reached from vertex 2 are :-
1, 2, 3, 4,

******************************************************************************/
```

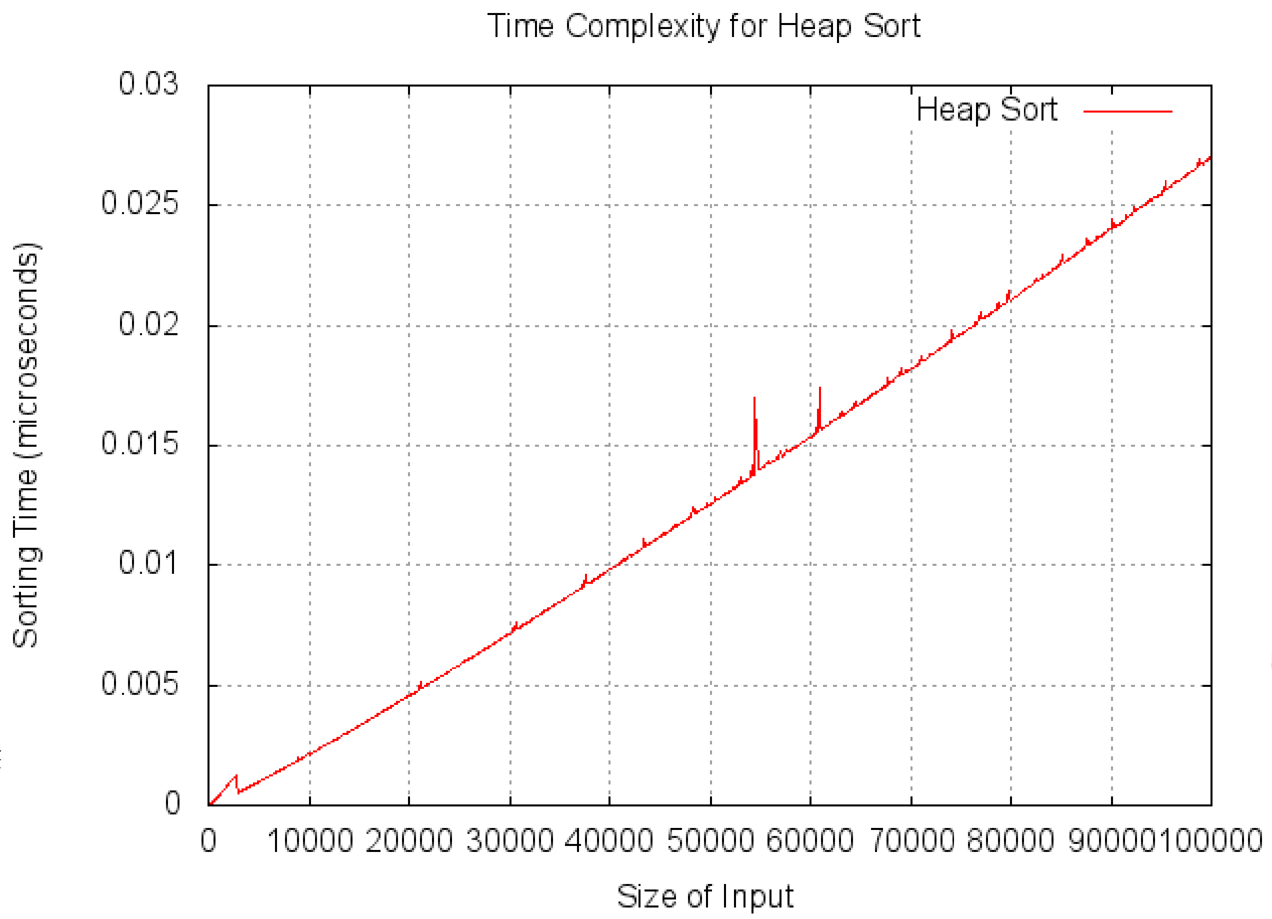
Question 6

Sort a given set of elements using the Heap sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and 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.

```
/******************************************************************************
*File       : HeapSort.c
*Description      : Program to sort an array using Heap Sort
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : Friday 22 November 2013
******************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

void fnGenRandInput(int [], int);
void fnDispArray( int [], int);
void fnMaxHeapify(int[],int,int);
void fnBuildMaxHeap(int[],int);
void fnHeapSort(int[],int);

/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{

      FILE *fp;
      struct timeval tv;
      double dStart,dEnd;
      int iaArr[500000],iNum,iPos,iKey,i,iChoice;

    for(;;)
    {
        printf("\n1.Plot the Graph\n2.HeapSort\n3.Exit");
        printf("\nEnter your choice\n");
        scanf("%d",&iChoice);

        switch(iChoice)
        {
            case 1:
                fp = fopen("HeapPlot.dat","w");

                for(i=100;i<100000;i+=100)
                {
                    fnGenRandInput(iaArr,i);

                    gettimeofday(&tv,NULL);
                    dStart = tv.tv_sec + (tv.tv_usec/1000000.0);

                fnHeapSort(iaArr,i);
```

```
                        gettimeofday(&tv,NULL);
                        dEnd = tv.tv_sec + (tv.tv_usec/1000000.0);

                        fprintf(fp,"%d\t%lf\n",i,dEnd-dStart);

                    }
                    fclose(fp);

                    printf("\nData File generated and stored in file < HeapPlot.dat
>.\n Use a plotting utility\n");
                break;

                case 2:
                    printf("\nEnter the number of elements to sort\n");
                    scanf("%d",&iNum);
                    printf("\nUnsorted Array\n");
                    fnGenRandInput(iaArr,iNum);
                    fnDispArray(iaArr,iNum);
                fnHeapSort(iaArr,iNum);
                    printf("\nSorted Array\n");
                    fnDispArray(iaArr,iNum);
                break;

                case 3:
                    exit(0);
            }
        }

      return 0;
}


/******************************************************************************
*Function   : fnMaxHeapify
*Description      : Function to recreate a max heap
*Input parameters:
*     int a[] - Array to hold integers
*     int i - start index of the Array
*     int n - end index of the Array
*RETURNS    : no value
******************************************************************************/

void fnMaxHeapify(int a[],int i,int n)
{
      int l,r,largest,temp;
      l=2*i;
      r=(2*i)+1;
      if((l<=n)&& (a[l]>a[i]))
      {
            largest=l;
      }
      else
      largest=i;
      if((r<=n)&&(a[r]>a[largest]))
      largest=r;
      if(largest!=i)
      {
            temp=a[i];
            a[i]=a[largest];
            a[largest]=temp;
            fnMaxHeapify(a,largest,n);
      }
}


/*******************************************************************************
*Function   : fnBuildMaxHeap
*Description      : Function to create a max heap
*Input parameters:
*     int a[] - Array to hold integers
*     int n - number of elements in the Array
*RETURNS    : no value
*******************************************************************************/

void fnBuildMaxHeap(int a[],int n)
{
      int i;
      for(i=n/2;i>=1;i--)
      fnMaxHeapify(a,i,n);
}

/*******************************************************************************
*Function   : fnHeapSort
*Description      : Function to sort an array using Heap Sort
*Input parameters:
*     int a[] - Array to hold integers
*     int n - number of elements in the Array
*RETURNS    : no value
*******************************************************************************/
void fnHeapSort(int a[],int n)
{
      int temp,i;
      fnBuildMaxHeap(a,n);
      for(i=n;i>=2;i--)
      {
            temp=a[1];
            a[1]=a[i];
            a[i]=temp;
            fnMaxHeapify(a,1,i-1);
      }
}

/*******************************************************************************
*Function   : GenRandInput
*Description      : Function to generate a fixed number of random elements
*Input parameters:
*     int X[] - array to hold integers
*     int n - no of elements in the array
*RETURNS    :no return value
*******************************************************************************/

void fnGenRandInput(int X[], int n)
{
      int i;

      srand(time(NULL));
      for(i=1;i<=n;i++)
      {
            X[i] = rand()%10000;
      }
}
```

```
/*************************************************************************************
*Function    : DispArray
*Description      : Function to display elements of an array
*Input parameters:
*      int X[] - array to hold integers
*      int n - no of elements in the array
*RETURNS     : no return value
************************************************************************************/

void fnDispArray( int X[], int n)
{
      int i;
      for(i=1;i<=n;i++)
            printf(" %5d \n",X[i]);
}
/*************************************************************************************
HeapPlot.gpl

# Gnuplot script file for plotting data in file "HeapPlot.dat"
# This file is called       HeapPlot.gpl
set terminal png font arial
set title "Time Complexity for Heap Sort"
set autoscale
set xlabel "Size of Input"
set ylabel "Sorting Time (microseconds)"
set grid
set output "HeapPlot.png"
plot "HeapPlot.dat" t 'Heap Sort' with lines

Execution
$gnuplot HeapPlot.gpl
*************************************************************************************/
```



Question 7a

Implement Horspool algorithm for String Matching.

```
/***************************************************************************
*File       : Horspool.c
*Description: Program to find a matching sub string in a given text using
                  Horspool's Algorithm
*Author     : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : 21 Nov 2013
***************************************************************************/
#include <stdio.h>
#include <string.h>

const int MAX = 256;

int fnHorspool(char string[], char pattern[],int []);
void fnGenShiftTable(char *,int []);

/***************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
***************************************************************************/
int main(void)
{
      char text[MAX];
      char pattern[MAX];
      int shiftTable[MAX];
      int found;
      puts("Enter the source string : ");
      gets(text);
      puts("Enter the pattern string : ");
      gets(pattern);

      fnGenShiftTable(pattern,shiftTable);
      found = fnHorspool(text,pattern,shiftTable);

      if(found==-1)
            puts("\nMatching Substring not found.\n");
      else
            printf("\nMatching Substring found at position: %d\n",found+1);

      return 0;
}
/***************************************************************************
*Function   : fnGenShiftTable
*Description      : Function to generate the shift table
*Input parameters:
*     char p[] - pattern to be searched for
*     int t[]     - shift table containing shift values for each alphabet
*RETURNS    : no value
***************************************************************************/
void fnGenShiftTable(char p[], int t[])
{
      int m, i, j;

      m = strlen(p);
      for(i=0; i<MAX; i++)
      {
            t[i]=m;
      }
      for(j=0; j<m-1; j++)
      {
            t[p[j]] = m-1-j;
      }
}

/*******************************************************************************
*Function   : fnHorspool
*Description      : Function to search for a matching substring for a given
                  pattern in the text
*Input parameters:
*     char s[] - text string
*     char p[] - pattern to be searched for
*     int t[]     - shift table containing shift values for each alphabet
*RETURNS    : no value
*******************************************************************************/
int fnHorspool(char s[],char p[],int t[])
{
      int i, n, m, k;

      n = strlen(s);
      m = strlen(p);
      i = m-1;
      while(i<n)
      {
            k = 0;
            while((k<m)&&(p[m-1-k]==s[i-k]))
                  k++;

            if (k == m)
                  return i-m+1;
            else
                  i = i+t[s[i]];
      }

      return -1;
}
/*******************************************************************************
Sample 1:
Enter the source string :
I want you to rebel!
Enter the pattern string :
you

Matching Substring found at position: 8

Sample 2:
Enter the source string :
Be a rebel!
Enter the pattern string :
coward

Matching Substring not found.

*******************************************************************************/
```

Question 7b

For a given set of elements construct an AVL Tree and also display balance factor for each node.

```
/******************************************************************************
*File       : AVL.c
*Description: Program to construct an AVL Tree for a given set of elements
*                      and also display balance factor for each node.
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : Saturday 14 December 2013
******************************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>

struct node
{
      int info,balance;
      struct node *lchild,*rchild;
};

typedef struct node * NODEPTR;

void inorder(NODEPTR);
NODEPTR insert (int , NODEPTR, bool *);
NODEPTR search(NODEPTR,int);

/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{
      bool ht_inc;
      int info,choice;
      NODEPTR root;
      root = NULL;

      while(1)
      {
            printf("1.Insert\n2.Display\n3.Quit\nEnter your choice : ");
            scanf("%d",&choice);
            switch(choice)
            {
                  case 1:printf("Enter the value to be inserted : ");
                        scanf("%d", &info);
                        if( search(root,info) == NULL )
                              root=insert(info, root, &ht_inc);
                        else
                              printf("Duplicate value ignored\n");
                        break;
                  case 2:if(root==NULL)
                        {
                              printf("Tree is empty\n");
                              continue;
                        }
                        printf("Tree is :\n");
                        printf("Inorder Traversal is: ");
                        inorder(root);
                        printf("\n \n");
                        break;
                case 3:exit(1);
                default:printf("Wrong choice\n");
            }
        }
        return 0;
}
/*******************************************************************************
*Function          : search
*Description       : Function to search for a given node in a AVL tree
*Input parameters:
*     NODEPTR pptr - pointer to root node
*     int info     - element to be searched
*RETURNS           : pointer to a node
*******************************************************************************/
NODEPTR search(NODEPTR ptr,int info)
{
        if(ptr!=NULL)
            if(info < ptr->info)
                ptr=search(ptr->lchild,info);
            else if( info > ptr->info)
                ptr=search(ptr->rchild,info);
        return(ptr);
}
/*******************************************************************************
*Function          : insert
*Description       : Function to insert an element into a AVL tree
*Input parameters:
*     int info     - element to be inserted
*     NODEPTR pptr - pointer to parent node
*     bool *ht_inc - boolean value
*RETURNS           : pointer to a node
*******************************************************************************/
NODEPTR insert (int info, NODEPTR pptr, bool *ht_inc)
{
        NODEPTR aptr,bptr;
        if(pptr==NULL)
        {
            pptr = (NODEPTR) malloc(sizeof(struct node));
            pptr->info = info;
            pptr->lchild = NULL;
            pptr->rchild = NULL;
            pptr->balance = 0;
            *ht_inc = true;
            return (pptr);
        }
        if(info < pptr->info)
        {
            pptr->lchild = insert(info, pptr->lchild,ht_inc);
            if(*ht_inc==true)
            {
                switch(pptr->balance)
                {
                        case -1: /* Right heavy */
                        pptr->balance = 0;
                        *ht_inc = false;
                        break;
                        case 0: /* Balanced */
                        pptr->balance = 1;
                        break;
                        case 1: /* Left heavy */
                        aptr = pptr->lchild;
                        if(aptr->balance == 1)
                        {
                              printf("\nR Rotation performed during
Insertion\n\n");
                              pptr->lchild= aptr->rchild;
                              aptr->rchild = pptr;
                              pptr->balance = 0;
                              aptr->balance=0;
                              pptr = aptr;
                        }
                        else
                        {
                              printf("\nLR Rotation performed during
Insertion\n\n");
                              bptr = aptr->rchild;
                              aptr->rchild = bptr->lchild;
                              bptr->lchild = aptr;
                              pptr->lchild = bptr->rchild;
                              bptr->rchild = pptr;
                              if(bptr->balance == 1 )
                                    pptr->balance = -1;
                              else
                                    pptr->balance = 0;
                              if(bptr->balance == -1)
                                    aptr->balance = 1;
                              else
                                    aptr->balance = 0;
                              bptr->balance=0;
                              pptr=bptr;
                        }
                        *ht_inc = false;
                  }
            }
      }
      if(info > pptr->info)
      {
            pptr->rchild = insert(info, pptr->rchild,ht_inc);
            if(*ht_inc==true)
            {
                  switch(pptr->balance)
                  {
                  case 1: /* Left heavy */
                        pptr->balance = 0;
                        *ht_inc = false;
                        break;
                  case 0: /* Balanced */
                        pptr->balance = -1;
                        break;
                  case -1: /* Right heavy */
                        aptr = pptr->rchild;
                        if(aptr->balance == -1)
                        {
                        printf("\nL Rotation performed during Insertion\n\n");
                        pptr->rchild= aptr->lchild;
                        aptr->lchild = pptr;
                        pptr->balance = 0;
                        aptr->balance=0;
                        pptr = aptr;
                        }
                        else
                        {
                        printf("\nRL Rotation performed during Insertion\n\n");
                        bptr = aptr->lchild;
                        aptr->lchild = bptr->rchild;
                        bptr->rchild = aptr;
                        pptr->rchild = bptr->lchild;
                        bptr->lchild = pptr;
                        if(bptr->balance == -1)
                              pptr->balance = 1;
                        else
                              pptr->balance = 0;
                        if(bptr->balance == 1)
                              aptr->balance = -1;
                        else
                              aptr->balance = 0;
                        bptr->balance=0;
                        pptr = bptr;
                        }
                        *ht_inc = false;
                  }
            }
      }
      return(pptr);
}

/********************************************************************************
*Function         : inorder
*Description      : Function to display inorder traversal of AVL tree with
*                           balance factor at each node
*Input parameters:
*     NODEPTR pptr - pointer to root node
*RETURNS          : nothing
********************************************************************************/

void inorder(NODEPTR ptr)
{
      if(ptr!=NULL)
      {
            inorder(ptr->lchild);
            printf("%d(%d) ",ptr->info,ptr->balance);
            inorder(ptr->rchild);
      }
}
```

Question 8a

Implement 0/1 Knapsack problem using dynamic programming.

```
/******************************************************************************
*File       : knapSack.cpp
*Description: Program to solve 0/1 Knapsack problem using dynamic programming.
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : 11 Nov 2013
******************************************************************************/
#include <iostream>
#include <cstdlib>
using namespace std;

const int MAX = 10;

inline int max(int a, int b);
void fnProfitTable(int w[MAX], int p[MAX], int n, int c, int t[MAX][MAX]);
void fnSelectItems(int n,int c, int t[MAX][MAX], int w[MAX], int l[MAX]);

/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{
      int i, j, totalProfit;
      int weight[MAX];
      int profit[MAX];
      int capacity;
      int num;
      int loaded[MAX];
      int table[MAX][MAX];


      cout << "Enter the maxium number of objects : ";
      cin >> num;

      cout << "Enter the weights : \n";
      for (i=1; i<=num; i++)
      {
          cout << "\nWeight " << i << ": ";
            cin >> weight[i];
    }
      cout << "\nEnter the profits : \n";
      for (i=1; i<=num; i++)
      {
          cout << "\nProfit " << i << ": ";
            cin >> profit[i];
    }
      cout << "\nEnter the maximum capacity : ";
      cin >> capacity;


      totalProfit = 0;

      for( i=1; i<=num; i++)
            loaded[i] = 0;
      fnProfitTable(weight,profit,num,capacity,table);
      fnSelectItems(num,capacity,table,weight,loaded);
      cout << "Profit Matrix\n";
      for (i=0; i<=num; i++)
      {
          for(j=0; j<=capacity; j++)
          {
              cout <<"\t"<<table[i][j];
          }
          cout << endl;
      }

      cout << "\nItem numbers which are loaded : \n{ ";
      for (i=1; i<=num; i++)
      {
            if (loaded[i])
            {
                  cout << i << " ";
                  totalProfit += profit[i];
            }
      }
      cout << "}" << endl;

      cout << "\nTotal Profit : " << totalProfit << endl;
      return 0;
}

inline int max(int a, int b)
{
      return a>b ? a : b;
}

/******************************************************************************
*Function   : fnProfitTable
*Description      : Function to construct the profit table
*Input parameters:
*     int w[MAX]  -     weight vector
*     int p[MAX]  -     profit vector
*     int n - no of items
*     int c - knapsack capacity
*     int t[MAX][MAX] - profit table
*RETURNS    : no value
******************************************************************************/
void fnProfitTable(int w[MAX], int p[MAX], int n, int c, int t[MAX][MAX])
{
      int i,j;

      for (j=0; j<=c; j++)
            t[0][j] = 0;

      for (i=0; i<=n; i++)
            t[i][0] = 0;

      for (i=1; i<=n; i++)
      {
            for (j=1; j<=c; j++)
            {
                  if (j-w[i] < 0)
                        t[i][j] = t[i-1][j];
                  else
                        t[i][j] = max( t[i-1][j], p[i] + t[i-1][j-w[i]]);
            }
      }
}
/********************************************************************************
*Function   : fnSelectItems
*Description: Function to determine optimal subset that fits into the knapsack
*Input parameters:
*     int n - no of items
*     int c - knapsack capacity
*     int t[MAX][MAX] - profit table
*     int w[MAX]  -     weight vector
*     int l[MAX]  -     bit vector representing the optimal subset
*RETURNS    : no value
********************************************************************************/
void fnSelectItems(int n,int c, int t[MAX][MAX], int w[MAX], int l[MAX])
{
      int i,j;

      i = n;
      j = c;
      while (i >= 1 && j >= 1)
      {
                  if (t[i][j] != t[i-1][j])
                  {
                        l[i] = 1;
                        j = j - w[i];
                        i--;
                  }
                  else
                        i--;
      }
}
/********************************************************************************
Enter the maxium number of objects : 4
Enter the weights :

Weight 1: 2
Weight 2: 1
Weight 3: 3
Weight 4: 2

Enter the profits :

Profit 1: 12
Profit 2: 10
Profit 3: 20
Profit 4: 15

Enter the maximum capacity : 5
Profit Matrix
      0     0     0     0     0     0
      0     0     12    12    12    12
      0     10    12    22    22    22
      0     10    12    22    30    32
      0     10    15    25    30    37

Item numbers which are loaded :
{ 1 2 4 }

Total Profit : 37
********************************************************************************/
```

Question 8b

Compute the transitive closure of a given directed graph using Warshall's algorithm.

```
/******************************************************************************
*File       : Warshall.c
*Description      : Program to find transitive closure of a given directed
*                           graph using Warshall's algorithm.
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : 11 Nov 2013
******************************************************************************/
#include <stdio.h>
const int MAX = 100;

void WarshallTransitiveClosure(int graph[MAX][MAX], int numVert);
/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{
      int i, j, numVert;
      int graph[MAX][MAX];

      printf("Warshall's Transitive Closure\n");
      printf("Enter the number of vertices : ");
      scanf("%d",&numVert);

      printf("Enter the adjacency matrix :-\n");
      for (i=0; i<numVert; i++)
            for (j=0; j<numVert; j++)
                  scanf("%d",&graph[i][j]);

      WarshallTransitiveClosure(graph, numVert);

      printf("\nThe transitive closure for the given graph is :-\n");
      for (i=0; i<numVert; i++)
      {
            for (j=0; j<numVert; j++)
            {
                  printf("%d\t",graph[i][j]);
            }
            printf("\n");
      }

      return 0;
}

/******************************************************************************
*Function   : WarshallTransitiveClosure
*Description      : Function to transitive closure of a given directed
*                           graph using Warshall's algorithm.
*Input parameters:
*     int graph[MAX][MAX] - adjacency matrix of the input graph
*     int numVert - no of vertices in the graph
*RETURNS    : no value
******************************************************************************/

void WarshallTransitiveClosure(int graph[MAX][MAX], int numVert)
```

```
{
      int i,j,k;

      for (k=0; k<numVert; k++)
      {
            for (i=0; i<numVert; i++)
            {
                  for (j=0; j<numVert; j++)
                  {
                        if (graph[i][j] || (graph[i][k] && graph[k][j]))
                              graph[i][j] = 1;
                  }
            }
      }
}

/*******************************************************************************
Warshall's Transitive Closure
Enter the number of vertices : 4
Enter the adjacency matrix :-
0 0 1 0
0 0 0 1
1 0 0 0
0 1 0 0

The transitive closure for the given graph is :-
1     0     1     0
0     1     0     1
1     0     1     0
0     1     0     1


Warshall's Transitive Closure
Enter the number of vertices : 4
Enter the adjacency matrix :-

0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0

The transitive closure for the given graph is :-
1     1     1     1
1     1     1     1
1     1     1     1
1     1     1     1
******************************************************************************/
```

Question 9a

Implement All Pair Shortest paths problem using Floyd's algorithm.

```
/***************************************************************************
*File       : Floyd.c
*Description: Program to implement Floyd's Algorithm
*Author     : Prabodh C P
*Compiler   : gcc compiler, Ubuntu 12.04
*Date       : 20 November 2013
***************************************************************************/

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

/******************************************************************************
*Function    : main
*Input parameters: no parameters
*RETURNS     :     0 on success
******************************************************************************/
int main(void)
{
      int iN, i, j, k;
      int iaFloyd[10][10], iaCost[10][10];

      printf("\n*************************************************************");
      printf("\n*\tPROGRAM TO IMPLEMENT FLOYD'S ALGORITHM\t*\n");
      printf("*************************************************************");

      printf("\nEnter the iNber of vertices\n");
      scanf("%d",&iN);

      printf("\nEnter the Cost adjacency Matrix\n");
      for(i=0;i<iN;i++)
            for(j=0;j<iN;j++)
                  scanf("%d",&iaCost[i][j]);

      printf("\nInput Graph\n");

      for(i=0;i<iN;i++)
      {
            for(j=0;j<iN;j++)
            {
                  printf("%d\t",iaCost[i][j]);
            }
            printf("\n");
      }
      printf("\n");

      for(i=0;i<iN;i++)
      {
            for(j=0;j<iN;j++)
            {
                  iaFloyd[i][j] = iaCost[i][j];
            }
      }

      for(k=0;k<iN;k++)
      {
            for(i=0;i<iN;i++)
            {
                  for(j=0;j<iN;j++)
                  {
                        iaFloyd[i][j] = (iaFloyd[i][j] < (iaFloyd[i][k] +
iaFloyd[k][j]))? (iaFloyd[i][j]):(iaFloyd[i][k] + iaFloyd[k][j]);
                  }
            }
      }
      printf("\nAll Pair Shortest Path Matrix\n");

      for(i=0;i<iN;i++)
      {
            for(j=0;j<iN;j++)
            {
                  printf("%d\t",iaFloyd[i][j]);
            }
            printf("\n");
      }
      printf("\n");

      return 0;
}
/****************************************************************************
Compiling instructions
$ gcc filename.c -o FLOYD.x
Execution
$./FLOYD.x

OUTPUT
SAMPLE 1

Input Graph
0     9999  3     9999
2     0     9999  9999
9999  7     0     1
6     9999  9999  0

All Pair Shortest Path Matrix
0     10    3     4
2     0     5     6
7     7     0     1
6     16    9     0

SAMPLE 2

Input Graph

0     15    10    9999  45    9999
9999  0     15    9999  20    9999
20    9999  0     20    9999  9999
9999  10    9999  0     35    9999
9999  9999  9999  30    0     9999
9999  9999  9999  4     9999  0

All Pair Shortest Path Matrix
0     15    10    30    35    9999
35    0     15    35    20    9999
20    30    0     20    50    9999
45    10    25    0     30    9999
75    40    55    30    0     9999
49    14    29    4     34    0
****************************************************************************/
```

Question 9b

Find the Binomial Co-efficient using Dynamic Programming.

```
/
******************************************************************************
*
*File       : BinomialCoeff.cpp
*Description      : Program to find Binomial Coefficient
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : Friday 22 November 2013
******************************************************************************
*/
#include <iostream>
using namespace std;

const int MAXSIZE = 10;

int fnBinomialCoefficient(int n, int k);

/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{
      int n, k;

      cout << "Binomial Coefficients\n";
      cout << "Calculates the value of nCk\n";
      cout << "Enter the value of n : ";
      cin >> n;
      cout << "Enter the value of k : ";
      cin >> k;

      if (n<k)
            cout << "\nInvalid Input n should be larger than k!\n";
      else
            cout << "\nThe value of " << n << "C" << k << " is " <<
fnBinomialCoefficient(n, k) << endl;

    return 0;
}
/******************************************************************************
*Function   : fnBinomialCoefficient
*Description      : Function to find Binomial Coefficient
*Input parameters:
*     int n - no of elements
*     int k - no of elements chosen out of n elements
*RETURNS    : value of nCk
******************************************************************************/
int fnBinomialCoefficient(int n, int k)
{
      int c[MAXSIZE][MAXSIZE];
      int i,j;

      for (i=0; i<=n; i++)
      {
            c[i][0] = 1;
            c[i][i] = 1;
      }

      for (i=1; i<=n; i++)
            for (j=1; j<i; j++)
                  c[i][j] = c[i-1][j-1] + c[i-1][j];

      cout << "\nThe Binomial matrix is :" << endl << "\t";
      for (i=0; i<=n; i++)
            cout << i << "\t";
      cout << endl;

      for (i=0; i<=n+1; i++)
            cout<<"========";
      cout << endl;
      for (i=0; i<=n; i++)
      {
            cout << "i=" << i << "\t";
            for (j=0; j<=i; j++)
            {
                  cout << c[i][j] << "\t";
            }
            cout << endl;
      }
      cout << endl;

      return c[n][k];
}

/********************************************************************************
Sample 1:
Binomial Coefficients
Calculates the value of nCk
Enter the value of n : 4
Enter the value of k : 6

Invalid Input n should be larger than k!

Sample 2:
Binomial Coefficients
Calculates the value of nCk
Enter the value of n : 6
Enter the value of k : 2

The Binomial matrix is :
      0     1     2     3     4     5     6
================================================================
i=0   1
i=1   1     1
i=2   1     2     1
i=3   1     3     3     1
i=4   1     4     6     4     1
i=5   1     5     10    10    5     1
i=6   1     6     15    20    15    6     1


The value of 6C2 is 15

********************************************************************************/
```

Question 10

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

```
/******************************************************************************
*File       : Dijkstra.cpp
*Description      : Program to find shortest paths to other vertices
*                             using Dijkstra's algorithm.
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : Friday 22 November 2013
******************************************************************************/
#include <iostream>
#include <cstdio>
using namespace std;

const int MAXNODES = 10,INF = 9999;

void fnDijkstra(int [][MAXNODES], int [], int [], int[], int, int, int);

/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{
    int n,cost[MAXNODES]
[MAXNODES],dist[MAXNODES],visited[MAXNODES],path[MAXNODES],i,j,source,dest;

    cout << "\nEnter the number of nodes\n";
    cin >> n;
    cout << "Enter the Cost Matrix\n" << endl;
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            cin >> cost[i][j];

//    cout << "Enter the Source vertex" << endl;
//    cin >> source;
//    cout << "Enter the Destination vertex" << endl;
//    cin >> dest;

    for (source = 0; source < n; source++)
    {
      getchar();
      cout << "\n//For Source Vertex : " << source  << " shortest path to other
vertices//"<< endl;
        for (dest=0; dest < n; dest++)
        {
            fnDijkstra(cost,dist,path,visited,source,dest,n);


            if (dist[dest] == INF)
                cout << dest << " not reachable" << endl;
            else
            {
                cout << endl;
                i = dest;
            do
                {
                    cout << i << "<--";
                    i = path[i];
                }while (i!= source);
                cout << i << " = " << dist[dest] << endl;
            }
        }
      cout << "Press Enter to continue...";
    }

    return 0;
}

/********************************************************************************
*Function   : fnDijkstra
*Description      : Function to find shortest paths to other vertices
*                            using Dijkstra's algorithm.
*Input parameters:
*     int c[][] - cost adjacency matrix of the graph
*     int d[] - distance vector
*     int p[] - path vector
*     int s[] - vector to store visited information
*     int so      - source vertex
*     int de      - destination vertex
*     int n - no of vertices in the graph
*RETURNS    : no value
********************************************************************************/
void fnDijkstra(int c[MAXNODES][MAXNODES], int d[MAXNODES], int p[MAXNODES],
int s[MAXNODES], int so, int de, int n)
{
    int i,j,a,b,min;

    for (i=0;i<n;i++)
    {
        s[i] = 0;
        d[i] = c[so][i];
        p[i] = so;
    }

    s[so] = 1;

    for (i=1;i<n;i++)
    {
        min = INF;
        a = -1;
        for (j=0;j<n;j++)
        {
            if (s[j] == 0)
            {
                if (d[j] < min)
                {
                    min = d[j];
                    a = j;
                }
            }
        }

        if (a == -1) return;

        s[a] = 1;

        if (a == de) return;
        for (b=0;b<n;b++)
        {
            if (s[b] == 0)
            {
                if (d[a] + c[a][b] < d[b])
                {
                    d[b] = d[a] + c[a][b];
                    p[b] = a;
                }
            }
        }
    }
}

/********************************************************************************
Enter the number of nodes
5
Enter the Cost Matrix

0     3     9999  7     9999
3     0     4     2     9999
9999  4     0     5     6
7     2     5     0     4
9999  9999  6     4     0

//For Source Vertex : 0 shortest path to other vertices//

0<--0 = 0

1<--0 = 3

2<--1<--0 = 7

3<--1<--0 = 5

4<--3<--1<--0 = 9
Press Enter to continue...

//For Source Vertex : 1 shortest path to other vertices//

0<--1 = 3

1<--1 = 0

2<--1 = 4

3<--1 = 2

4<--3<--1 = 6
Press Enter to continue...

//For Source Vertex : 2 shortest path to other vertices//

0<--1<--2 = 7

1<--2 = 4

2<--2 = 0

3<--2 = 5
```

```
4<--2 = 6
Press Enter to continue...

//For Source Vertex : 3 shortest path to other vertices//

0<--1<--3 = 5

1<--3 = 2

2<--3 = 5

3<--3 = 0

4<--3 = 4
Press Enter to continue...

//For Source Vertex : 4 shortest path to other vertices//

0<--1<--3<--4 = 9

1<--3<--4 = 6

2<--4 = 6

3<--4 = 4

4<--4 = 0
Press Enter to continue...
*******************************************************************************/
```

Question 11

Find Minimum Cost Spanning Tree of a given undirected graph using Prims algorithm.

```
/******************************************************************************
*File       : Prim.cpp
*Description      : Program to find Minimum Cost Spanning Tree of a given
*                        undirected graph using Prim's algorithm.
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : Friday 22 November 2013
******************************************************************************/

#include <iostream>
using namespace std;

const int MAXNODES = 10;
void fnPrims(int n, int cost[MAXNODES][MAXNODES]);
void fnGetMatrix(int n,int a[MAXNODES][MAXNODES]);

/******************************************************************************
*Function   : main
*Input parameters:
*     int argc - no of commamd line arguments
*     char **argv - vector to store command line argumennts
*RETURNS    :     0 on success
******************************************************************************/
int main( int argc, char **argv)
{
      int a[MAXNODES][MAXNODES] = {       {0, 3, 9999, 7, 9999},
                        {3, 0, 4, 2, 9999},
                        {9999, 4, 0, 5, 6},
                        {7, 2, 5, 0, 4},
                        {9999, 9999, 6, 4, 0}};

      int n = 5;

      cout << "Enter the number of vertices : ";
      cin >> n;

      fnGetMatrix(n,a);
      fnPrims(n,a);

      return 0;
}

/******************************************************************************
*Function   : fnPrims
*Description      : Function to find Minimum Cost Spanning Tree of a given
*                        undirected graph using Prims algorithm.
*Input parameters:
*     int n - no of vertices in the graph
*     int cost[][] - cost adjacency matrix of the graph
*RETURNS    : no value
******************************************************************************/
void fnPrims(int n, int cost[MAXNODES][MAXNODES])
{
      int i, j, u, v, min;
      int sum, k, t[MAXNODES][2], p[MAXNODES], d[MAXNODES], s[MAXNODES];
      int source, count;
```

```
min = 9999;
source = 0;

for(i=0; i<n; i++)
{
      for(j=0; j<n; j++)
      {
            if(cost[i][j] != 0 && cost[i][j] <= min)
            {
                  min = cost[i][j];
                  source = i;
            }
      }
}

for(i=0; i<n; i++)
{
      d[i] = cost[source][i];
      s[i] = 0;
      p[i] = source;
}
s[source] = 1;
sum = 0;
k = 0;
count = 0;

while (count != n-1)
{
      min = 9999;
      u = -1;
      for(j=0; j<n; j++)
      {
            if(s[j] == 0)
            {
                  if(d[j] <= min)
                  {
                         min = d[j];
                         u = j;
                  }
            }
      }

      t[k][0] = u;
      t[k][1] = p[u];
      k++;
      count++;
      sum += cost[u][p[u]];
      s[u] = 1;

      for(v=0; v<n; v++)
      {
            if(s[v]==0 && cost[u][v]<d[v])
            {
                  d[v] = cost[u][v];
                  p[v] = u;
            }
      }
}

if(sum >= 9999)
      cout << "\nSpanning tree does not exist\n";
      else
      {
            cout << "\nThe spanning tree exists and minimum cost spanning tree
is \n";
            for(i=0; i<k; i++)
                  cout << t[i][1] << " " << t[i][0] << endl;

            cout << "\nThe cost of the minimum cost spanning tree is " << sum
<< endl;
      }
}

/********************************************************************************
*Function   : fnGetMatrix
*Description      : Function to read cost adjacency matrix of the graph
*Input parameters:
*     int n - no of vertices in the graph
*     int a[][] - cost adjacency matrix of the graph
*RETURNS    : no value
********************************************************************************/
void fnGetMatrix(int n,int a[MAXNODES][MAXNODES])
{
      int i, j;

    cout << "Enter the Cost Adjacency Matrix" << endl;
      for(i=0; i<n; i++)
            for(j=0; j<n; j++)
                  cin >> a[i][j];
}

/********************************************************************************
Output Sample 2:
Enter the number of vertices : 5
Enter the Cost Adjacency Matrix

0     3     9999  7     9999
3     0     4     2     9999
9999  4     0     5     6
7     2     5     0     4
9999  9999  6     4     0

The spanning tree exists and minimum cost spanning tree is
3 1
1 0
3 4
1 2

The cost of the minimum cost spanning tree is 13

Output Sample 2:
Enter the number of vertices : 5
Enter the Cost Adjacency Matrix

0     3     9999  7     9999
3     0     9999  2     9999
9999  9999  0     9999  9999
7     2     9999  0     4
9999  9999  9999  4     0

Spanning tree does not exist
********************************************************************************/
```

Question 12

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

```
/******************************************************************************
*File       : Kruskal.cpp
*Description      : Program to find Minimum Cost Spanning Tree of a given
*                       undirected graph using Kruskal's algorithm.
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : Friday 22 November 2013
******************************************************************************/

#include <iostream>
using namespace std;

const int MAXNODES = 10;
const int INF = 9999;

// Structure to represent an edge
struct edge
{
      int u, v, cost;
};

int fnFindParent(int v, int parent[]);
void fnUnion_ij(int i, int j, int parent[]);
void fnInputGraph(int m, edge e[]);
int fnGetMinEdge(edge e[], int n);
void kruskal(int n, edge e[], int m);

/******************************************************************************
*Function   : main
*Input parameters:
*     int argc - no of commamd line arguments
*     char **argv - vector to store command line argumennts
*RETURNS    :     0 on success
******************************************************************************/
int main( int argc, char **argv)
{
      int n = 6, m = 10;
      edge e[2*MAXNODES] = {{0,1,6},{1,4,3},{4,5,6},{5,3,2},{3,0,5},{0,2,1},
{1,2,5},{3,2,5},{4,2,6},{5,2,4}};

      cout << "Enter the number of nodes : ";
      cin >> n;
      cout << "Enter the number of edges : ";
      cin >> m;

      fnInputGraph(m, e);
      kruskal(n, e, m);

      return 0;
}

/******************************************************************************
*Function   : fnFindParent
*Description      : Function to find parent of a given vertex
*Input parameters:
*     int v - vertex for whom parent has to be found
*     int parent[] - parent vector
*RETURNS    : parent vertex
********************************************************************************/

int fnFindParent(int v, int parent[])
{
      while (parent[v] != v)
            v = parent[v];

      return v;
}

/********************************************************************************
*Function   : fnUnion_ij
*Description      : Function to merge two trees
*Input parameters:
*     int i, j - vertices to be merged
*     int parent[] - parent vector
*RETURNS    : no value
********************************************************************************/

void fnUnion_ij(int i, int j, int parent[])
{
      if(i < j)
            parent[j] = i;
      else
            parent[i] = j;
}
/********************************************************************************
*Function   : fnInputGraph
*Description      : Function to read a graph
*Input parameters:
*     int m - no of edges in the graph
*     edge e[] - set of edges in the graph
*RETURNS    : no value
********************************************************************************/
void fnInputGraph(int m, edge e[])
{
      int i, j, k, cost;

      for(k=0; k<m; k++)
      {
            cout << "Enter edge and cost in the form u, v, w : \n";
            cin >> i >> j >> cost;

            e[k].u = i;
            e[k].v = j;
            e[k].cost = cost;
      }
}

/********************************************************************************
*Function   : fnGetMinEdge(
*Description      : Function to find the least cost edge in the edge set
*Input parameters:
*     edge e[] - set of edges in the graph
*     int n - no of vertices in the graph
*RETURNS    : index of least cost edge in the edge set
********************************************************************************/
int fnGetMinEdge(edge e[], int n)
{
	int i, small, pos;
	small = INF;
	pos = -1;

	for(i=0; i<n; i++)
	{
		if(e[i].cost < small)
		{
			small = e[i].cost;
			pos = i;
		}
	}

	return pos;
}

void kruskal(int n, edge e[], int m)
{
	int i, j, count, k, sum, u, v, t[MAXNODES][2], pos;
	int parent[MAXNODES];
	count = 0;
	k = 0;
	sum = 0;

	for(i=0; i<n; i++)
	{
		parent[i] = i;
	}

	while(count != n-1)
	{
		pos = fnGetMinEdge(e,m);
		if(pos == -1)
		{
			break;
		}
		u = e[pos].u;
		v = e[pos].v;
		i = fnFindParent(u,parent);
		j = fnFindParent(v,parent);

		if(i != j)
		{
			t[k][0] = u;
			t[k][1] = v;
			k++;
			count++;
			sum += e[pos].cost;
			fnUnion_ij(i, j, parent);
		}
		e[pos].cost = INF;
	}

	if(count == n-1)
	{
		cout << "\nSpanning tree exists";
		cout << "\nThe Spanning tree is shown below\n";
		for(i=0; i<n-1; i++)
			cout << t[i][0] << " " << t[i][1] << endl;

		cout << "\nCost of the spanning tree : " << sum;
```

```
        }
        else
              cout << "\nThe spanning tree does not exist";
}

/********************************************************************************
Enter the number of nodes : 6
Enter the number of edges : 10
Enter edge and cost in the form u, v, w :
0 1 6
Enter edge and cost in the form u, v, w :
1 4 3
Enter edge and cost in the form u, v, w :
4 5 6
Enter edge and cost in the form u, v, w :
5 3 2
Enter edge and cost in the form u, v, w :
3 0 5
Enter edge and cost in the form u, v, w :
0 2 1
Enter edge and cost in the form u, v, w :
1 2 5
Enter edge and cost in the form u, v, w :
3 2 5
Enter edge and cost in the form u, v, w :
4 2 6
Enter edge and cost in the form u, v, w :
5 2 4

Spanning tree exists
The Spanning tree is shown below
0 2
5 3
1 4
5 2
1 2

Cost of the spanning tree : 15
********************************************************************************/
```

Question 13

Find a subset of a given set S = {s_1,s_2,.....,s_n} of n positive integers whose sum is equal to a given positive integer d. 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 doesn't have a solution.

```
/******************************************************************************
*File       : Subset.cpp
*Description: Program to solve Subset sum problem.
*Author           : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : 11 Nov 2013
******************************************************************************/
#include <iostream>
using namespace std;

// Constant definitions
const int MAX = 100;

// class definitions
class SubSet
{
      int stk[MAX], set[MAX];
      int size, top, count;
      public:
      SubSet()
      {
            top = -1;
            count = 0;
      }
      void getInfo(void);
      void push(int data);
      void pop(void);
      void display(void);
      int fnFindSubset(int pos, int sum);
};

/******************************************************************************
*Function   : getInfo
*Description: Function to read input
*Input parameters: no parameters
*RETURNS    : no value
******************************************************************************/

void SubSet :: getInfo(void)
{
      int i;
      cout << "Enter the maximum number of elements : ";
      cin >> size;

      cout << "Enter the weights of the elements : \n";
      for (i=1; i<=size; i++)
            cin >> set[i];

}
/******************************************************************************
*Function   : push
*Description: Function to push an element on to the stack
*Input parameters:
```

```
*int data   - value to be pushed on to the stack
*RETURNS    : no value
******************************************************************************/
void SubSet :: push(int data)
{
      stk[++top] = data;
}

/******************************************************************************
*Function   : pop
*Description: Function to pop an element from the stack
*Input parameters: no parameters
*RETURNS    : no value
******************************************************************************/

void SubSet :: pop(void)
{
      top--;
}

/******************************************************************************
*Function   : display
*Description: Function to display solution to sub set sum problem
*Input parameters: no parameters
*RETURNS    : no value
******************************************************************************/
void SubSet :: display()
{
      int i;
      cout << "\nSOLUTION #"<< ++count <<" IS\n{ ";
      for (i=0; i<=top; i++)
            cout << stk[i] << " ";

      cout << "}" << endl;
}

/******************************************************************************
*Function   : fnFindSubset
*Description      : Function to solve Subset sum problem.
*Input parameters:
*     int pos     - position
*     int sum     - sum of elements
*RETURNS    : returns 1 if solution exists or zero otherwise
******************************************************************************/
int SubSet :: fnFindSubset(int pos, int sum)
{
      int i;
      static int foundSoln = 0;

      if (sum>0)
      {
            for (i=pos; i<=size; i++)
            {
                  push(set[i]);
                  fnFindSubset(i+1, sum - set[i]);
                  pop();
            }
      }

      if (sum == 0)
      {
            display();
            foundSoln = 1;
      }
      return foundSoln;
}
/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{
      int i,sum;

      SubSet set1;

      set1.getInfo();
      cout << "Enter the total required weight : ";
      cin >> sum;

      cout << endl;

      if (!set1.fnFindSubset(1, sum))
            cout << "\n\nThe given problem instance doesnt have any solution."
<< endl;
      else
            cout << "\n\nThe above-mentioned sets are the required solution to
the given instance." << endl;

      return 0;
}

/******************************************************************************
OUTPUT
SAMPLE 1

Enter the maximum number of elements : 5
Enter the weights of the elements :
1 2 3 4 5
Enter the total required weight : 5

SOLUTION #1 IS
{ 1 4 }

SOLUTION #2 IS
{ 2 3 }

SOLUTION #3 IS
{ 5 }

The above-mentioned sets are the required solution to the given instance.

SAMPLE 2

Enter the maximum number of elements : 4
Enter the weights of the elements :
1 2 3 4
Enter the total required weight : 11

The given problem instance doesnt have any solution.
******************************************************************************/
```

Question 14

Implement N Queen's problem using Back Tracking.

```
/******************************************************************************
*File       : nQueens.cpp
*Description: Program to solve N Queens problem using backtracking.
*Author          : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : 11 Nov 2013
******************************************************************************/
#include <iostream>
#include <cstdlib>
using namespace std;

const int MAX = 10;

int SolnCount =0;

void fnChessBoardShow(int n, int row[MAX]);
bool fnCheckPlace(int KthQueen, int ColNum, int row[MAX]);
int NQueen(int k,int n, int row[MAX]);

/******************************************************************************
*Function   : main
*Input parameters: no parameters
*RETURNS    :     0 on success
******************************************************************************/
int main(void)
{
      int n;
      int row[MAX];
      cout << "Enter the number of queens : ";
      cin >> n;

      if (!NQueen(0,n,row))
            cout << "No solution exists for the given problem instance." <<
endl;
      else
            cout << "Number of solution for the given problem instance is : "
<< SolnCount << endl;

      return 0;
}
/******************************************************************************
*Function   : NQueen
*Description      : Function to place n queens on a nxn chess board without any
*                             queen attacking any other queen
*Input parameters:
*     int k -     kth queen
*     int n - no of queens
*     int row[MAX] - vector containing column numbers of each queen
*RETURNS    : returns 1 if solution exists or zero otherwise
******************************************************************************/

int NQueen(int k,int n, int row[MAX])
{
      static int flag;
      for(int i=0; i<n; i++)
      {
            if(fnCheckPlace(k,i,row) == true)
            {
                  row[k] = i;
                  if(k == n-1)
                  {
                        fnChessBoardShow(n,row);
                        SolnCount++;
                        flag = 1;
                        return flag;
                  }
                  NQueen(k+1, n, row);
            }
      }
      return flag;
}
/******************************************************************************
*Function   : fnCheckPlace
*Description: Function to check whether a kth queen can be palced in a specific
*                             column or not
*Input parameters:
*     int KthQueen      -     kth queen
*     int ColNum        - columnn number
*     int row[MAX]      - vector containing column numbers of each queen
*RETURNS    : returns true if the queen can be palced or false otherwise
******************************************************************************/
bool fnCheckPlace(int KthQueen, int ColNum, int row[MAX])
{
      for(int i=0; i<KthQueen; i++)
      {
            if(row[i] == ColNum || abs(row[i]-ColNum) == abs(i-KthQueen))
                  return false;
      }

      return true;
}

/******************************************************************************
*Function   : fnChessBoardShow
*Description: Function to graphically display solution to n queens problem
*Input parameters:
*     int n - no of queens
*     int row[MAX]      - vector containing column numbers of each queen
*RETURNS    : no value
******************************************************************************/

void fnChessBoardShow(int n, int row[MAX])
{
      cout << "\nSolution #" << SolnCount+1 << endl << endl;

      for (int i=0; i<n; i++)
      {
            for (int j=0; j<n; j++)
            {
                  if (j == row[i])
                        cout << "Q ";
               else
                        cout << "# ";
            }
            cout << endl;
      }
      cout << endl;
```

```
}

/*****************************************************************************
OUTPUT
SAMPLE 1

Enter the number of queens : 4

Solution #1

# Q # #
# # # Q
Q # # #
# # Q #


Solution #2

# # Q #
Q # # #
# # # Q
# Q # #

Number of solution for the given problem instance is : 2

SAMPLE 2

Enter the number of queens : 3
No solution exists for the given problem instance.

*****************************************************************************/
```