B.M.S. EDUCATION TRUST

B.M.S.COLLEGE OF ENGINEERING, BANGALORE-19

(Autonomous College under VTU)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY MANUAL

15CS5DCADA

PROGRAM: BACHELOR OF ENGINEERING

SEMESTER: V

COURSECODE: 15CS5DCADA

COURSE TITLE: ANALYSIS AND DESIGN OF ALGORITHMS

CREDITS: 5

PREFACE

This laboratory manual is prepared by the Department of Computer Science and engineering for Analysis and Design of Algorithms (15CS5DCADA). This lab manual can be used as instructional book for students, staff and instructors to assist in performing and understanding the experiments. In this manual, experiments as per syllabus are described.

Instructions to Students to be followed in each ADA lab:

1. Each Student should write down the work carried out and the outputs in the observation book and get it evaluated by the respective lab faculty in-charge.
2. Each Student should bring the lab record with the programs and output written for the programs completed in their respective previous week and get it evaluated by the lab faculty in-charge. In the record book students should
- Handwrite the Algorithm
- Handwrite the Program
- Pasting of the printout of the Output and Graph or Handwriting of the Output and Graph.

3. Each Student should practice the extra exercises given in each lab.

Note: Students after completion of the Labwork
- Desktop system used should be ShutDown
- Should put back the used Keyboard, Mouse and Chair properly before leaving the lab.

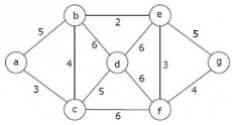
Design, develop and implement the specified algorithms for the following problems using C Language in **LINUX** / Windows environment. But preferably on LINUX environment.

Prog.#	Lab	Name of Program	Exercise to Complete
0	0	Write a recursive program to a. Solve Towers-of-Hanoi problem b. To find GCD	Write a recursive program to a. Generate N Fibonacci numbers c. Find Factorial of a given number
1	1	Implement Recursive **Binary search** and **Linear search** and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N. Note: In the record book students should - Handwrite the Algorithm - Handwrite the Program - Pasting of the printout of the Output and Graph or Handwrite of	Write program to sort the numbers using **Bubble sort**. Repeat experiment for different values of N and plot the graph.

		the Output and Graph.	
2	1	Sort a given set of N integer elements using **Selection Sort** technique and compute its time taken. Run the program for different values of N and record the time taken to sort. Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator. Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output and Graph or handwrite of the Output and Graph.	
3	2	Sort a given set of N integer elements using **Merge Sort technique** and compute its time taken. Run the program for different values of N and record the time taken to sort. Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator. Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output and Graph or handwrite of the Output and Graph.	Note down the time taken by Bubble sort (carried out in previous week lab). Compare Bubble sort with Merge sort and Quick sort. Write your comments in your observation book regarding comparison of time taken between these three algorithms.
4	2	Sort a given set of N integer elements using **Quick Sort** technique and compute its time taken. Run the program for different values of N and record the time taken to sort. Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator. Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output and Graph or handwriting of the Output and Graph.	

5	3	Write program to do the following: **a.** Print all the nodes reachable from a given starting node in a digraph using **BFS** method. **b.** Check whether a given graph is connected or not using **DFS** method. Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output or Handwrite of the Output .	**a.** Write a program using BFS to check whether the given graph is cyclic or not **b.** Write a program using DFS to check whether the given graph is cyclic or not
6	3	Write program to obtain the **Topological ordering** of vertices in a given digraph. Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output or Handwrite of the Output	
7	4	Sort a given set of N integer elements using **Insertion Sort** technique and compute its time taken. Run the program for different values of N and record the time taken to sort. Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator. Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output and Graph or handwrite the Output and Graph.	- Modify the program such that the given input array elements should be ascending order using Insertion sort - Compare the time taken between two test cases Test case 1: The given input of the array elements is in ascending order Test case 2: The given input of the array elements is in random - Note down the time taken by Bubble sort (carried out in previous week lab), and Insertion sort. Write your comments in your observation book regarding comparison of time taken between these two algorithms.
8	5	Implement **"Sum of Subsets"** using Backtracking.	In your observation book write state-space tree for Subset-sum problem where S={3,5,6,7} and

		"Sum of Subsets" problem: Find a subset of a given set S = {s1,s2,……,sn} 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. Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output or handwrite the Output	d=15
9	6	Implement **"N-Queens Problem"** using Backtracking. Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output or handwrite the Output	In your observation book write state-space tree for Four-Queens problem
10	7	Write program to find the **Binomial Co-efficient** using Dynamic Programming. Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output or handwrite the Output	
11	7	Implement **0/1 Knapsack problem** using dynamic programming. Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output or handwrite the Output	
12	8	Implement All Pair Shortest paths problem using **Floyd's algorithm.**	Compare the time taken by various

		Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output or handwrite the Output	sorting algorithms such as Bubble sort, Selection sort, Insertion sort, Quick sort, Merge sort and Heap sort. Do the comparison for different test cases such that the given input array elements are in Ascending order, Descending order and Random. In your observation book write your comments
13	8	Sort a given set of N integer elements using **Heap Sort** technique and compute its time taken. Run the program for different values of N and record the time taken to sort. Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator. Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output and Graph or handwrite the Output and Graph.	
14	9	Find Minimum Cost Spanning Tree of a given undirected graph using **Prim's algorithm.** Note: In the record book students should - Handwrite the Algorithm, - Handwrite the Program - Pasting of the printout of the Output or handwrite the Output	In your observation write down the sequence of edges added to the minimum spanning tree by applying Prims and Krushkals algorithm. Cross check your manually generated output with program output. 
15	9	Find Minimum Cost Spanning Tree of a given undirected graph using **Kruskals algorithm**. Note: In the record book students should - Handwrite the Algorithm - Handwrite the Program - Pasting of the printout of the Output or handwrite the Output	
16	10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using **Dijkstra's algorithm**. Note: In the record book students should	

		- Handwrite the Algorithm - Handwrite the Program - Pasting of the printout of the Output or handwrite the Output	

General Description:

a. Computing time taken

To compute the time taken by the algorithm, we make use of built-in data type clock_t which is available in the header file time.h. Two variables of type clock_t are required to get the starting time and ending time of the algorithm. The built-in function clock() gives the current time of the system in terms of number of clock-ticks. So, to compute the time elapsed for the working of an algorithm, the different between starting time and ending time has to be considered and it must be divided by CLK_TCK or CLOCKS_PER_SEC.

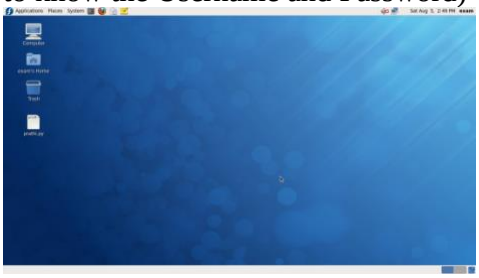
As the processors are very fast, the time computed will be negligibly small and hence it may result in zero always. Hence, a delay is enforced wherever the basic operation is expected to execute.

Normally, time complexity can be better understood with a huge value of input size. Hence, instead of reading the array elements from the keyboard, it is suggested to generate random numbers

b. Running C program on Linux environment

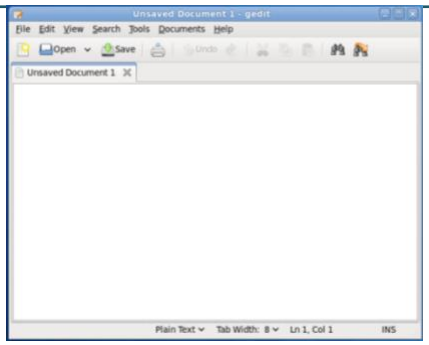
Following is Example screenshots for writing and executing C program on Linux environment

Step 1: Login into Fedora Username: ***** and Password: ******* (Ask lab Systems administrator to know the Username and Password)

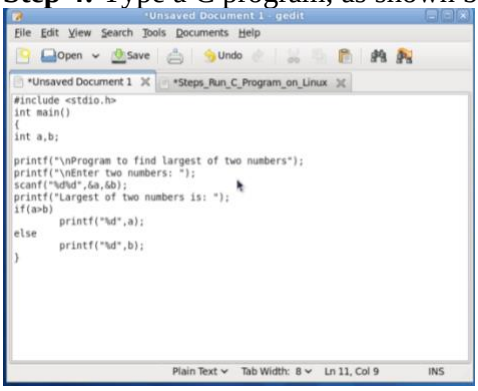


Step 2: Click on Applications -> Accessories -> gedit Text Editor

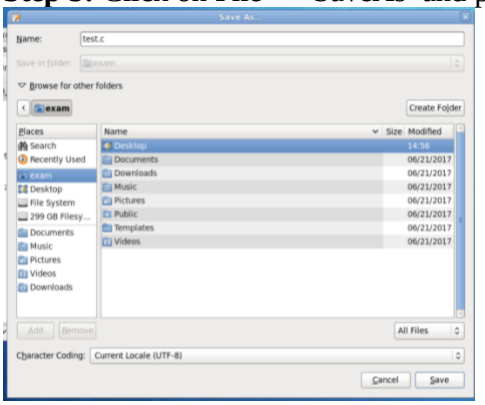
Step 3: You will see the screen as shown below



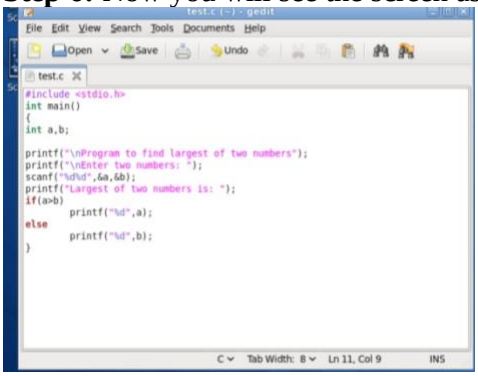
Step 4: Type a C program, as shown below



Step 5: Click on File -> SaveAs and provide filename as "test.c"



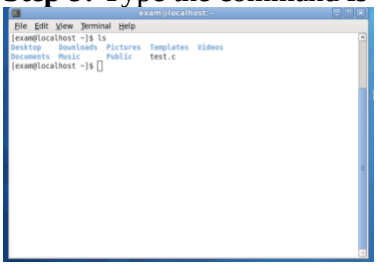
Step 6: Now you will see the screen as shown below



Step 7: Next, Click on Applications -> System Tools -> Terminal

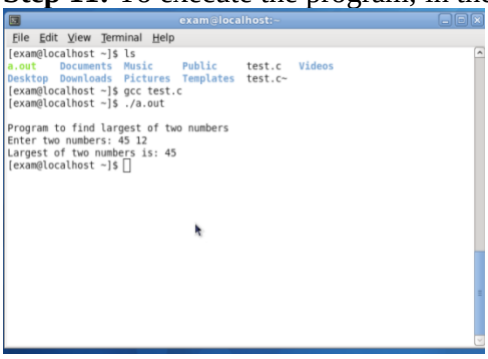
Step 8: Now you will see the Terminal screen as shown below

Step 9: Type the command **ls** . You will see your program listed



Step 10: To compile the program, in the terminal type the command ***gcc test.c***

Step 11: To execute the program, in the terminal type the command ***./a.out***

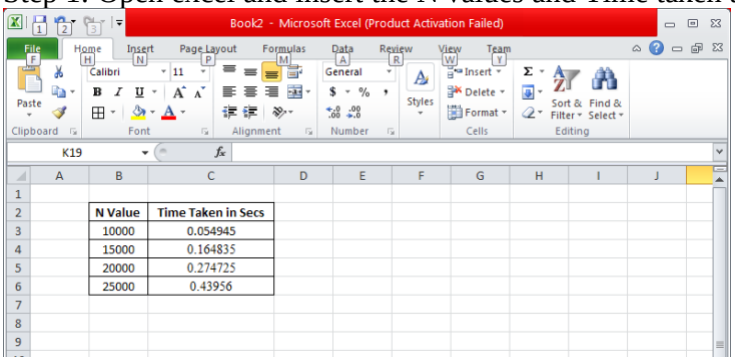


Step 12: To shutdown the system, Click on System -> Shutdown

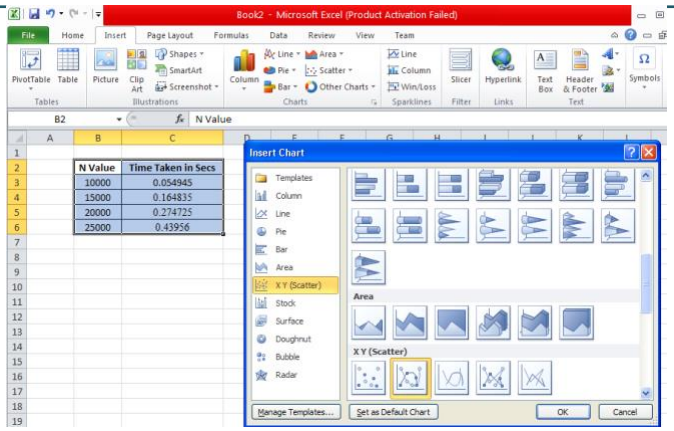
c. Plotting graph using MS Excel

Following is an Example to plot graph in MS Excel

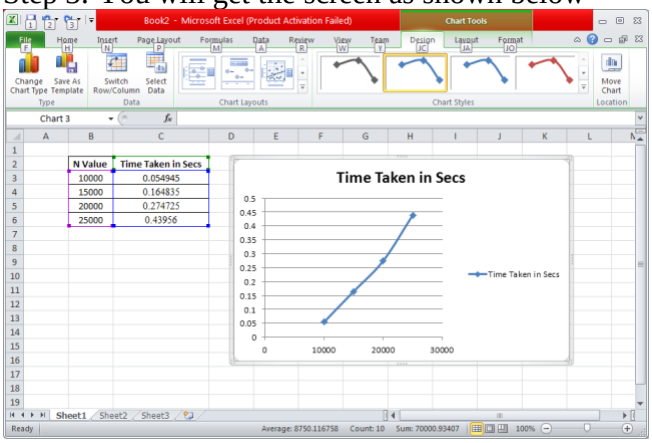
Step 1: Open excel and insert the N values and Time taken as shown below



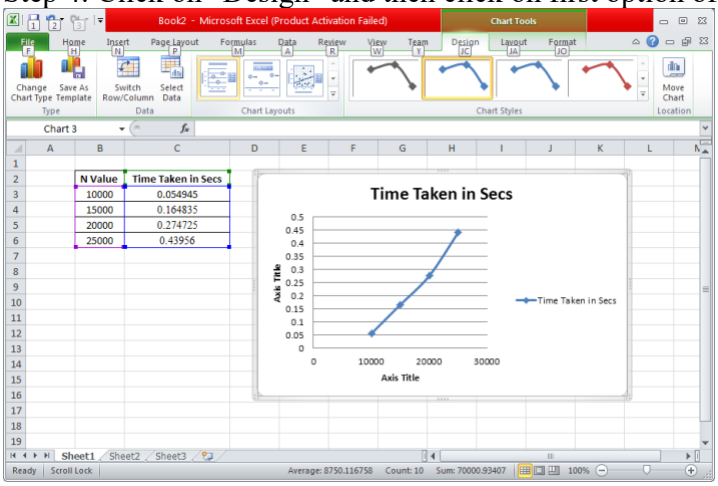
Step 2: Select the N values and Time taken then Click on “insert” and select “XY Scatter plot”, as shown below



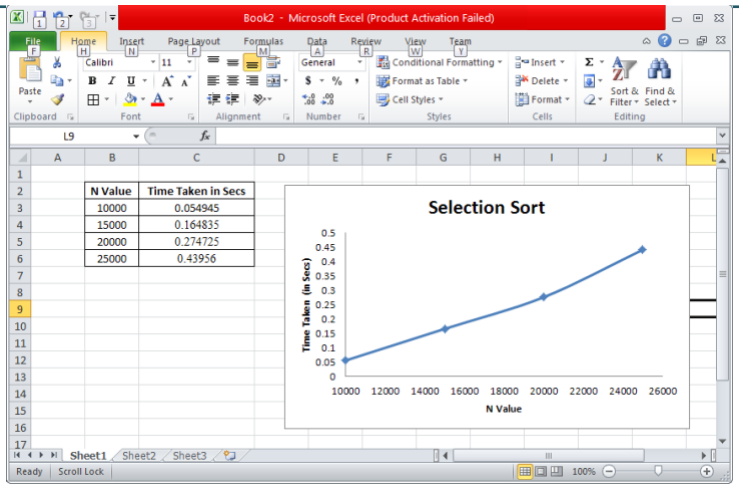
Step 3: You will get the screen as shown below



Step 4: Click on “Design” and then click on first option of the “Chart Layouts”



Step 5: Provide X-axis and Y-axis labels



N Value	Time Taken in Secs
10000	0.054945
15000	0.164835
20000	0.274725
25000	0.43956

Lab Zero:

Write a recursive program to a. Solve Towers-of-Hanoi problem b. To find GCD.

```
#include(stdio.h) // place this '<' & '>' instead of '(' & ')' before stdio.h
#include(conio.h)
#include(math.h)
void hanoi(int x, char from, char to, char aux)
{
if(x==1)
printf("Move Disk From %c to %c\n",from,to);
else
{
hanoi(x-1,from,aux,to);
printf("Move Disk From %c to %c\n",from,to);
hanoi(x-1,aux,to,from);
}
}
void main( )
{
int disk;
int moves;
clrscr();
printf("Enter the number of disks you want to play with:");
scanf("%d",&disk);
moves=pow(2,disk)-1;
printf("\nThe No of moves required is=%d \n",moves);
hanoi(disk,'A','C','B');
getch( );
}
```

Output:-

```
Enter the number of disks you want to play with: 3
The No of moves required is=7
Move Disk from A to C
Move Disk from A to B
Move Disk from C to B
Move Disk from A to C
Move Disk from B to A
Move Disk from B to C
Move Disk from A to C
```

```
#include <stdio.h>
int hcf(int n1, int n2);
int main()
{
  int n1, n2;
  printf("Enter two positive integers: ");
  scanf("%d %d", &n1, &n2);

  printf("G.C.D of %d and %d is %d.", n1, n2, hcf(n1,n2));
  return 0;
}

int hcf(int n1, int n2)
{
   if (n2 != 0)
      return hcf(n2, n1%n2);
   else
      return n1;
}
```

EXPERIMENT 1

AIM: Implement Recursive **Binary search** and **Linear search** and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N.
Note: In the record book students should
- Handwrite the Algorithm
- Handwrite the Program
- Pasting of the printout of the Output and Graph or handwriting of the Output and Graph.

Algorithm:

```
bin_srch(a[0….n-1],key)
//Implements recursive binary search
//Input: An array a[0….n-1] sorted in ascending order
//          key- element to be searched
//Output: position of the array's element that is equal to key is returned otherwise -1 is returned.
low ←0; high←n-1
if low>high
```

```
        return -1
    end if
    mid←(low+high)/2
    if key = a[mid]
    return mid
    end if
    if key < a[mid]
      high←mid-1
    else
    low←mid+1
    end if
ALGORITHM :  lin_srch(a[0….n-1],key)
    //Implements sequential search with a search key as a sentinel
    //Input: An array a[0….n-1] sorted in ascending order
    //          key- element to be searched
    //Output: position of the first element in a[0….n-1] whose value is equal to key is returned otherwise -1 is
    //             returned.
    i←0; high←n-1
    if i>high
       return -1
    end if
            if key = a[i]
       return i
    else
      return lin_search(a,i+1,high,key)
    end if
```

Program:

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with gcc*/
int bin_srch(int [],int,int,int);
int lin_srch(int [],int,int,int);
void bub_sort(int[],int);
int n,a[10000];
int main()
{
 int ch,key,search_status,temp;
 clock_t end,start;
unsigned long int i, j;

 while(1)
 {
  printf("\n1: Binary search\t 2: Linear search\t 3: Exit\n");
  printf("\nEnter your choice:\t");
  scanf("%d",&ch);
  switch(ch)
  {
   case 1:
```

```
    n=1000;
          while(n<=5000)
          {
           for(i=0;i<n;i++)
           {
           //a[i]=random(1000);
           a[i]=i;  //Insering numbers in Ascending order
           }
           key=a[n-1]; //Last element of the aray
           start=clock();
       //bub_sort(a,n); //Sorting numbers in Ascending order using Bubble sort
           search_status=bin_srch(a,0,n-1,key);
           if(search_status==-1)
          printf("\nKey Not Found");
       else
          printf("\n Key found at position %d",search_status);
       //Dummy loop to create delay
           for(j=0;j<500000;j++){ temp=38/600;}
           end=clock();
           printf("\nTime for n=%d is %f Secs",n,(((double)(end-start))/CLOCKS_PER_SEC));
           n=n+1000;
          }
          break;
case  2:
      n=1000;
          while(n<=5000)
          {
       for(i=0;i<n;i++)
           {
            //a[i]=random(10000);
            a[i]=i;
           }
           key=a[n-1]; //Last element of the aray
           start=clock();
           search_status=lin_srch(a,0,n-1,key);
           if(search_status==-1)
          printf("\nKey Not Found");
       else
          printf("\n Key found at position %d",search_status);
       //Dummy loop to create delay
         for(j=0;j<500000;j++){ temp=38/600;}
           end=clock();
           printf("\nTime for n=%d is %f Secs",n,(((double)(end-start))/CLOCKS_PER_SEC));
           n=n+1000;
          }
          break;
```

```
   default:
          exit(0);
  }
  getchar();
 }
}
void bub_sort(int a[],int n)
{
 int i,j,temp;
 for(i=0;i<=n-2;i++)
 {
  for(j=0;j<=n-2-i;j++)
  {
   if(a[j]>a[j+1])
   {
    temp=a[j];
    a[j]=a[j+1];
    a[j+1]=temp;
   }
  }
 }
}
int bin_srch(int a[],int low,int high,int key)
{
 int mid;
 if(low>high)
 {
  return -1;
 }
 mid=(low+high)/2;
 if(key==a[mid])
 {
  return mid;
 }
 if(key<a[mid])
 {
  return bin_srch(a,low,mid-1,key);
 }
 else
 {
  return bin_srch(a,mid+1,high,key);
 }
}

int lin_srch(int a[],int i,int high,int key)
{
```

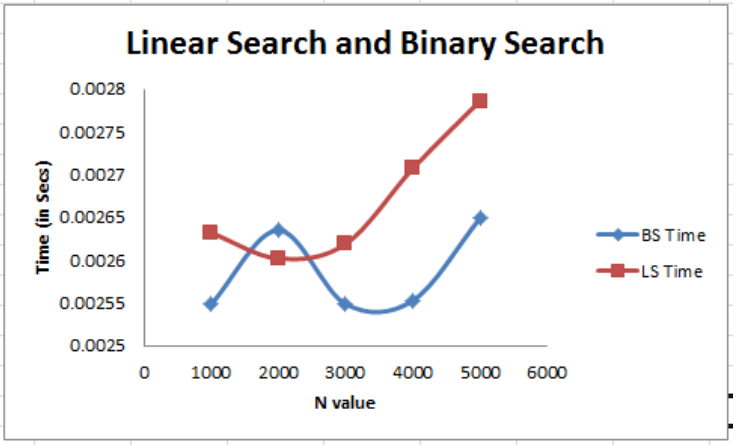
```
 if(i>high)
 {
  return -1;
 }
 if(key==a[i])
 {
  return i;
 }
 else
 {
  return lin_srch(a,i+1,high,key);
 }
}
```

Output:

```
1: Binary search        2: Linear search        3: Exit

Enter your choice:1
 Key found at position 999
Time for n=1000 is 0.002549 Secs
 Key found at position 1999
Time for n=2000 is 0.002635 Secs
 Key found at position 2999
Time for n=3000 is 0.002549 Secs
 Key found at position 3999
Time for n=4000 is 0.002553 Secs
 Key found at position 4999
Time for n=5000 is 0.002649 Secs

1: Binary search        2: Linear search        3: Exit
Enter your choice:2
 Key found at position 999
Time for n=1000 is 0.002632 Secs
 Key found at position 1999
Time for n=2000 is 0.002602 Secs
 Key found at position 2999
Time for n=3000 is 0.002619 Secs
 Key found at position 3999
Time for n=4000 is 0.002707 Secs
 Key found at position 4999
Time for n=5000 is 0.002786 Secs

1: Binary search        2: Linear search        3: Exit
Enter your choice: 3
```

Graph screenshot: It can be observed from graph below that time taken by Linear search is more when compared to Binary Search.

N Value	BS Time	LS Time
1000	0.002549	0.002632
2000	0.002635	0.002602
3000	0.002549	0.002619
4000	0.002553	0.002707
5000	0.002649	0.002786



EXPERIMENT 2

AIM: Sort a given set of N integer elements using **Selection Sort** technique and compute its time taken. Run the program for different values of N and record the time taken to sort.
Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator. Note: In the record book students should
- Handwrite the Algorithm
- Handwrite the Program
- Pasting of the printout of the Output and Graph or Handwriting of the Output and Graph.

Note: N value should be in the range

ALGORITHM : sel_sort(a[0….n-1]
```
//Sorts a given array by selection sort
//Input : An array a[0….n-1] of orderable elements
//Output : Array a[0….n-1] sorted in ascending order
for i←0 to n-2 do
    small_pos←i
        for j←i+1 to n-1 do
            if a[j]<a[small_pos]
                small_pos←j
            end if
        end for
    swap a[i] and a[small_pos]
end for
```

Program:

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with gcc*/
void selsort(int n,int a[]);

void main()
```

```
{
  int a[15000],n,i,j,ch,temp;
  clock_t start,end;

  while(1)
  {
printf("\n1:For manual entry of N value and array elements");
printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");
printf("\n3:To exit");
   printf("\nEnter your choice:");
   scanf("%d", &ch);
   switch(ch)
   {
     case 1:  printf("\nEnter the number of elements: ");
                  scanf("%d",&n);
                  printf("\nEnter array elements: ");
                  for(i=0;i<n;i++)
                   {
                    scanf("%d",&a[i]);
                   }
                  start=clock();
                  selsort(n,a);
                  end=clock();
                  printf("\nSorted array is: ");
                  for(i=0;i<n;i++)
                  printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
                  break;
    case 2:
              n=500;
              while(n<=14500) {
              for(i=0;i<n;i++)
                    {
                     //a[i]=random(1000);
                     a[i]=n-i;
                    }
              start=clock();
              selsort(n,a);
        //Dummy loop to create delay
            for(j=0;j<500000;j++){ temp=38/600;}
           end=clock();
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
                   n=n+1000;
                    }
```

```
                break;
   case 3: exit(0);
   }
   getchar();
    }
}


void selsort(int n,int a[])
{
    int i,j,t,small,pos;
    for(i=0;i<n-1;i++)
     {
      pos=i;
      small=a[i];
      for(j=i+1;j<n;j++)
      {
            if(a[j]<small)
            {
              small=a[j];
              pos=j;
            }
      }
      t=a[i];
      a[i]=a[pos];
      a[pos]=t;
   }
}
```

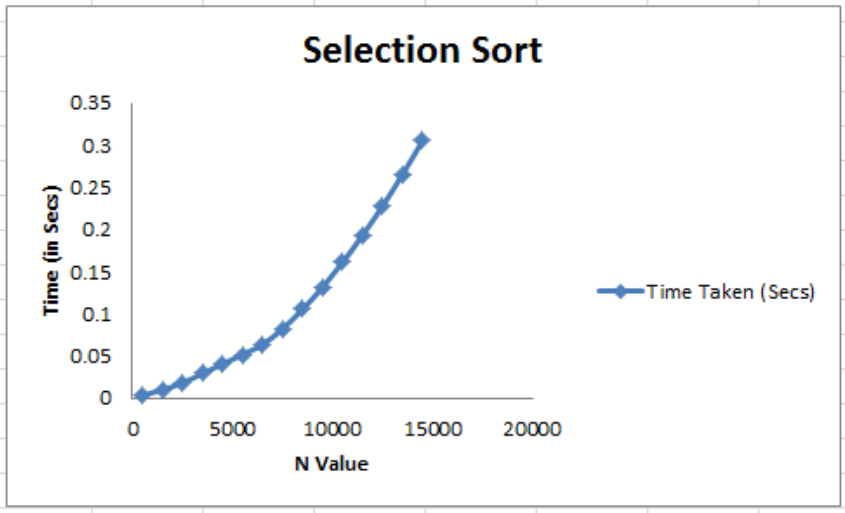
Output:

```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 10000 to 25000
3:To exit
Enter your choice:1
Enter the number of elements: 4
Enter array elements: 44 33 22 11
Sorted array is: 11      22      33      44
 Time taken to sort 4 numbers is 0.000001 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:2
 Time taken to sort 500 numbers is 0.003295 Secs
 Time taken to sort 1500 numbers is 0.010222 Secs
 Time taken to sort 2500 numbers is 0.018430 Secs
 Time taken to sort 3500 numbers is 0.029586 Secs
 Time taken to sort 4500 numbers is 0.040886 Secs
```

Time taken to sort 5500 numbers is 0.051843 Secs
Time taken to sort 6500 numbers is 0.063836 Secs
Time taken to sort 7500 numbers is 0.082561 Secs
Time taken to sort 8500 numbers is 0.105688 Secs
Time taken to sort 9500 numbers is 0.131809 Secs
Time taken to sort 10500 numbers is 0.161104 Secs
Time taken to sort 11500 numbers is 0.193703 Secs
Time taken to sort 12500 numbers is 0.227940 Secs
Time taken to sort 13500 numbers is 0.265653 Secs
Time taken to sort 14500 numbers is 0.305674 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 10000 to 25000
3:To exit
Enter your choice:3

Graph Screenshot:

	N value	Time Taken (Secs)
2	500	0.003295
3	1500	0.010222
4	2500	0.01843
5	3500	0.029586
6	4500	0.040886
7	5500	0.051843
8	6500	0.063836
9	7500	0.082561
10	8500	0.105688
11	9500	0.131809
12	10500	0.161104
13	11500	0.193703
14	12500	0.22794
15	13500	0.265653
16	14500	0.305674



Graph Generation:
To be carried out by Students: Generate graph for N value vs Time taken for comparison of Bubble Sort and Selection sort

EXPERIMENT 3

AIM: Sort a given set of N integer elements using **Merge Sort** technique and compute its time taken. Run the program for different values of N and record the time taken to sort. Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator.
Note: In the record book students should
- Handwrite the Algorithm,
- Handwrite the Program
- Pasting of the printout of the Output and Graph or handwriting of the Output and Graph.

ALGORITHM : combine(a[0….n-1],low,mid,high)

```
//merge two sorted arrays where first array starts from low to mid and second starts from mid+1 to high
//Input : a is a sorted array from index position low to mid
//        a is a sorted array from index position mid+1 to high
//Output: Array a[0….n-1] sorted in nondecreasing order
i←low
j←mid+1
k←low
while i<=mid and j<=high do
     if a[i]<a[j]
        c[k]←a[i]
         k←k+1
         i←i+1
     else
        c[k]←a[j]
        k←k+1
        j←j+1
     end if
end while
if i>mid
    while j<=high do
        c[k]←a[j]
        k←k+1
        j←j+1
    end while
end if
if j>high
    while i<=mid do
        c[k]←a[i]
        k←k+1
        i←i+1
    end while
end if
for i←low to high do
    a[i]←c[i]
end for

ALGORITHM: split(a[0….n-1],low,high)
//Sorts array a[0….n-1] by recursive mergesort
//Input :An array a[0….n-1] of orderable elements
//Output : Array a[0….n-1] sorted in nondecreasing order
if low<high
   mid←(low+high)/2
   split(a,low,mid)
   split(a,mid+1,high)
   combine(a,low,mid,high)
end if
```

Program:

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with gcc*/
void split(int[],int,int);
```

```
void combine(int[],int,int,int);
void main()
{
  int a[15000],n, i,j,ch, temp;
  clock_t start,end;

  while(1)
  {
 printf("\n1:For manual entry of N value and array elements");
 printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");
 printf("\n3:To exit");
    printf("\nEnter your choice:");
    scanf("%d", &ch);
    switch(ch)
    {
      case 1:  printf("\nEnter the number of elements: ");
                  scanf("%d",&n);
                  printf("\nEnter array elements: ");
                  for(i=0;i<n;i++)
                   {
                    scanf("%d",&a[i]);
                   }
                  start=clock();
                  split(a,0,n-1);
                  end=clock();
                  printf("\nSorted array is: ");
                  for(i=0;i<n;i++)
                  printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
                  break;
    case 2:
              n=500;
              while(n<=14500) {
              for(i=0;i<n;i++)
                    {
                     //a[i]=random(1000);
                     a[i]=n-i;
                    }
              start=clock();
              split(a,0,n-1);
        //Dummy loop to create delay
            for(j=0;j<500000;j++){ temp=38/600;}
              end=clock();
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
```

```
                        n=n+1000;
                         }
                    break;
   case 3: exit(0);
   }
   getchar();
    }
}

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

void combine(int a[],int low,int mid,int high)
{
 int c[15000],i,j,k;
 i=k=low;
 j=mid+1;
 while(i<=mid&&j<=high)
 {
  if(a[i]<a[j])
  {
   c[k]=a[i];
   ++k;
   ++i;
  }
  else
  {
   c[k]=a[j];
   ++k;
   ++j;
  }
 }
 if(i>mid)
 {
  while(j<=high)
  {
   c[k]=a[j];
```

```
  ++k;
  ++j;
 }
}
if(j>high)
{
 while(i<=mid)
 {
  c[k]=a[i];
  ++k;
  ++i;
 }
}
for(i=low;i<=high;i++)
{
 a[i]=c[i];
}
}
```

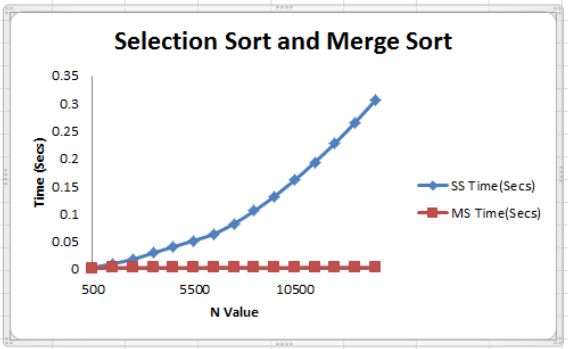
Output:

```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:1
Enter the number of elements: 4
Enter array elements: 44 33 22 11
Sorted array is: 11        22        33        44
 Time taken to sort 4 numbers is 0.000012 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:2
 Time taken to sort 500 numbers is 0.002698 Secs
 Time taken to sort 1500 numbers is 0.002907 Secs
 Time taken to sort 2500 numbers is 0.003055 Secs
 Time taken to sort 3500 numbers is 0.003391 Secs
 Time taken to sort 4500 numbers is 0.003037 Secs
 Time taken to sort 5500 numbers is 0.002826 Secs
 Time taken to sort 6500 numbers is 0.003703 Secs
 Time taken to sort 7500 numbers is 0.003794 Secs
 Time taken to sort 8500 numbers is 0.003021 Secs
 Time taken to sort 9500 numbers is 0.003072 Secs
 Time taken to sort 10500 numbers is 0.003144 Secs
 Time taken to sort 11500 numbers is 0.003546 Secs
 Time taken to sort 12500 numbers is 0.003321 Secs
 Time taken to sort 13500 numbers is 0.003507 Secs
```

Time taken to sort 14500 numbers is 0.003624 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:3
Graph Scree shot: It can be observed from the graph below that time taken by Selection sort is more when compared to Merge sort.

To be carried out by students: Write your observation in your book the reason for selection sort taking more time when compared to Merge sort.

N value	SS Time(Secs)	MS Time(Secs)
500	0.003295	0.002698
1500	0.010222	0.002907
2500	0.01843	0.003055
3500	0.029586	0.003391
4500	0.040886	0.003037
5500	0.051843	0.002826
6500	0.063836	0.003703
7500	0.082561	0.003794
8500	0.105688	0.00321
9500	0.131809	0.003072
10500	0.161104	0.003144
11500	0.193703	0.003546
12500	0.22794	0.003321
13500	0.265653	0.003507
14500	0.305674	0.003624



Graph Generation:

To be carried out by Students: Generate graph for N value vs Time taken for comparison of Bubble Sort, Selection sort and Merge sort

EXPERIMENT 4

AIM: Sort a given set of N integer elements using **Quick Sort** technique and compute its time taken. Run the program for different values of N and record the time taken to sort. Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator.
Note: In the record book students should
- Handwrite the Algorithm,
- Handwrite the Program
- Pasting of the printout of the Output and Graph or handwriting of the Output and Graph.

ALGORITHM: partition(a[0….n-1],low,high)
//partition the array into parts such that elements towards the left of the *key* element are
//less than *key* element and elements towards right of the *key* element are greater than *key* element
//Input: An array a[0….n-1] is unsorted from index position *low* to *high*
//Output: A partition of a[0…n-1] with split position returned as this function's value
key←a[low]
i←low+1
j←high
while(1)
    while a[i]<=key **and** i<=high **do**
        i←i+1
    end while
    while a[j]>key **and** j>=low **do**
        j←j-1
    end while
    if i<j

```
            swap a[i] and a[j]
        else
            swap a[low] and a[j]
            return j
        end if
end while
```

ALGORITHM: quick_sort(a[0….n-1],low,high)
//Sorts the elements of the array between lower bound *low* and upper bound *high*
//Input: An array a[0….n-1] is unsorted from index position *low* to *high*
//Output: Array a[0….n-1] is sorted in nondecreasing order

```
if low<high
   j←partition(a,low,high)
   quick_sort(a,low,j-1)
   quick_sort(a,j+1,high)
end if
```

Program: Will be provided shortly

EXPERIMENT 5

AIM: Write program to do the following:
a. Print all the nodes reachable from a given starting node in a digraph using **BFS** method.
b. Check whether a given graph is connected or not using **DFS** method.
Note: In the record book students should
- Handwrite the Algorithm,
- Handwrite the Program
- Pasting of the printout of the Output or Handwrite the Output .

ALGORITHM : bfs(a[1….n,1….n],src)
// Implements a breadth-first traversal of a given digraph
//Input: An adjacency matrix a[1….n,1….n] of given digraph
//src-from where the traversal is initiated
//Output: The digraph with its vertices marked with consecutive integers in the order they have been visited
//by the BFS traversal mark each vertex in vis[1….n] with 0 as mark of being "node is not reachable"

```
for j←1 to n do
    vis[j]←0
end for
f←0
r←-1
vis[src]←1
r←r+1
while f<=r do
     i←q[f]
     f←f+1
      for j←1 to n do
          if a[i,j]=1 and vis[j]!=1
              vis[j]←1
              r←r+1
              q[r]←j
          end if
      end for
end while
for j←1 to n do
   if vis[j]!=1
     write 'node is not reachable'
   else
```

```
      write 'node is reachable'
    end if
end for
```

ALGORITHM : dfs(a[1….n,1….n],src)

```
// Implements a depth-first traversal of a given digraph
//Input: An adjacency matrix a[1….n,1….n] of given digraph
//          src-from where the traversal is initiated
//Output: returns 0 if graph is not connected otherwise 1 is returned
vis[src]←1
for j←1 to n do
    if a[src,j]=1 and vis[j]!=1
       dfs(j)
    end if
end for
for j←1 to n do
    if vis[j]!=1
         write 'graph is not connected'
    end if
end for
write 'graph is connected'
return
```

Program:

```
#include<stdio.h>
#include<conio.h>
int a[10][10],n;
void bfs(int);
void main()
{
 int i,j,src;
 clrscr();
 printf("\nenter the no of nodes:\t");
 scanf("%d",&n);
 printf("\nenter the adjacency matrix:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   scanf("%d",&a[i][j]);
  }
 }
 printf("\nenter the source node:\t");
 scanf("%d",&src);
 bfs(src);
 getch();
}

void bfs(int src)
{
 int q[10],f=0,r=-1,vis[10],i,j;
 for(j=1;j<=n;j++)
 {
  vis[j]=0;
 }
 vis[src]=1;
 r=r+1;
 q[r]=src;
```

```
while(f<=r)
{
 i=q[f];
 f=f+1;
 for(j=1;j<=n;j++)
 {
  if(a[i][j]==1&&vis[j]!=1)
  {
   vis[j]=1;
   r=r+1;
   q[r]=j;
  }
 }
}
for(j=1;j<=n;j++)
{
 if(vis[j]!=1)
 {
  printf("\nnode %d is not reachable\n",j);
 }
 else
 {
  printf("\nnode %d is reachable\n",j);
 }
}
}
```

==========Output==========

```
Enter the no. of nodes:  6

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

Enter the source node:  1

Node  1  is reachable
Node  2  is reachable
Node  3  is reachable
Node  4  is reachable
Node  5  is reachable
Node  6  is reachable
========================
```

```
#include<stdio.h>
#include<conio.h>
int a[10][10],n,vis[10];
int dfs(int);
void main()
{
 int i,j,src,ans;
 clrscr();
 for(j=1;j<=n;j++)
 {
  vis[j]=0;
```

```
 }
 printf("\nenter the no of nodes:\t");
 scanf("%d",&n);
 printf("\nenter the adjacency matrix:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   scanf("%d",&a[i][j]);
  }
 }
 printf("\nenter the source node:\t");
 scanf("%d",&src);
 ans=dfs(src);
 if(ans==1)
 {
  printf("\ngraph is connected\n");
 }
 else
 {
  printf("\ngragh is not connected\n");
 }
 getch();
}

int dfs(int src)
{
 int j;
 vis[src]=1;
 for(j=1;j<=n;j++)
 {
  if(a[src][j]==1&&vis[j]!=1)
  {
   dfs(j);
  }
 }
 for(j=1;j<=n;j++)
 {
  if(vis[j]!=1)
  {
   return 0;
  }
 }
 return 1;
}
```

==========Output==========

```
Enter the no. of nodes:  4
Enter the adjacency matrix:
0   1   1   0
0   0   0   0
0   0   0   1
0   1   0   0
Enter the source node:  1
Graph is connected

Enter the no. of nodes:  4
Enter the adjacency matrix:
0   1   1   0
```

```
0  0  0  0
0  1  0  0
0  0  0  0

Enter the source node:  1
Graph is not connected
=========================
```

EXPERIMENT 6

AIM: Write program to obtain the Topological ordering of vertices in a given digraph.

Note: In the record book students should
- Handwrite the Algorithm,
- Handwrite the Program
- Pasting of the printout of the Output or Handwrite the Output

```
Algorithm topological_sort(a,n,T)
 //purpose :To obtain the sequence of jobs to be executed resulting topological order
// Input:a-adjacency matrix of the given graph
//          n-the number of vertices in the graph
//output:
// T-indicates the jobs that are to be executed in the order
Step 1:[Obtain indgree of each vertex]
For j <- 0 to n-1 do
            sum<-0
            for i<- 0 to n-1 do
                        sum<-sum+a[i]
            end for
            indegree[i] <- sum
end for
Step 2: [Place the independent jobs which have not been processed on the stack]
            For <- 0 to n-1 do
            If(indegree[i]=0)  //Place the job on the stack
                        top <- top+1
                        s[top] <- i
            end if
            end for
Step 3: [Find the topological sequence]
            While (top!=1)
                        u <- s[top]
                        top <- top -1
            Add u to solution vector T
            For each vertex v adjacent to u
                        Decrement indegree[v] by one
                        If(indegree[v]=0)
                                    Top <- top +1
                                    s[top] <- v
                        end if
            end for
            end while
Step 4: write T
Step 5: return
```

NOTE: The following program is developed for solving a topological ordering problem using source removal method. For this program, the graph should be directed acyclic graph. Hence, give the input adjacency matrix appropriately.

```
#include<stdio.h>
#include<conio.h>
void source_removal(int n, int a[10][10])
```

```
{
	inti,j,k,u,v,top,s[10],t[10],indeg[10],sum;
	for(i=0;i<n;i++)
	{
		sum=0;
		for(j=0;j<n;j++)
		{
			sum+=a[j][i];
		}
		indeg[i]=sum;
	}
	top=-1;
	for(i=0;i<n;i++)
	{
		if(indeg[i]==0)
		{
			s[++top]=i;
		}
	}
	k=0;
	while(top!=-1)
	{
		u=s[top--];
		t[k++]=u;
		for(v=0;v<n;v++)
		{
			if(a[u][v]==1)
			{
				indeg[v]=indeg[v]-1;
				if(indeg[v]==0)
					s[++top]=v;
			}
		}
	}
	for(i=0;i<n;i++)
	{
		printf("%d\n", t[i]);
	}
}
void main()
{
	inti,j,a[10][10],n;
	printf("Enter number of nodes\n");
	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]);
		}
	}
	source_removal(n,a);
	getch();
}
```

Output:

Enter number of nodes 6

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

1
4
0
2
3
5

EXPERIMENT 7

AIM: Sort a given set of N integer elements using **Insertion Sort** technique and compute its time taken. Run the program for different values of N and record the time taken to sort. Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator. Note: In the record book students should
- Handwrite the Algorithm
- Handwrite the Program
- Pasting of the printout of the Output and Graph or handwriting of the Output and Graph.

Algorithm: Step 1 − If it is the first element, it is already sorted. return 1;
Step 2 − Pick next element
Step 3 − Compare with all elements in the sorted sub-list
Step 4 − Shift all the elements in the sorted sub-list that is greater than the value to be sorted
Step 5 − Insert the value
Step 6 − Repeat until list is sorted

Program: **Will be provided shortly**

EXPERIMENT 8

AIM: Implement "Sum of Subsets" using Backtracking. "Sum of Subsets" problem: Find a subset of a given set S = {s1,s2,……,sn} 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.

ALGORITHM: subset(s[1….n],d)
// To find subsets of a given set of n positive integers whose sum is equal to a given positive integer *d*
//Input: An array s[1….n] of sorted elements
// d-required sum
//Output: subsets of given set s[1….n] whose elements sum is equal to *d*
x[k]←1
if m+s[k]=d
  write 'subset solution is', count←count+1
  for i←0 to k do
    if x[i]=1
      write s[i]
    end if
  end for
else if m+s[k]+s[k+1]<=d
    subset(m+s[k],k+1,sum-s[k])
end if
if m+sum-s[k]>=d and m+s[k+1]<=d

```
   x[k]←0
   subset(m,k+1,sum-s[k])
            end if
```

Program:

```
#include<stdio.h>
#include<conio.h>
intcount,w[10],d,x[10];
void subset(intcs, int k, int r)
{
int i;
x[k]=1;
if(cs+w[k]==d)
{
printf("\nSubset solution = %d\n", ++count);
for(i=0;i<=k;i++)
{
if(x[i]==1)
printf("%d", w[i]);
}
}
else
if(cs+w[k]+w[k+1]<=d)
subset(cs+w[k], k+1, r-w[k]);
if((cs+r-w[k]>=d) && (cs+w[k+1])<=d)
{
x[k]=0;
subset(cs,k+1,r-w[k]);
}
    }

void  main()
   {
int sum=0,i,n;
printf("Enter the number of elements\n");
scanf("%d", &n);
printf("Enter the elements in ascending     order\n");
for(i=0;i<n;i++)
scanf("%d", &w[i]);

printf("Enter the required sum\n");
scanf("%d", &d);
for(i=0;i<n;i++)
sum+=w[i];
if(sum<d)
{
printf("No solution exists\n");
return;
}
printf("The solution is\n");
count=0;
subset(0,0,sum);
getch();
}
```

Output:

```
Enter the number of elements
5
```

Enter the elements in ascending order
1
2
5
6
8
Enter the required sum
9
The solution is
Subset solution = 1
1 2 6
Subset solution = 2
1 8

EXPERIMENT 9

AIM: Implement "N-Queens Problem" using Backtracking.
Note: In the record book students should
- Handwrite the Algorithm,
- Handwrite the Program
- Pasting of the printout of the Output or handwrite the Output

ALGORITHM: nqueens(n)
//places n queens on a nXn matrix such that no two queens are placed along same row or same column
//or same diagonal using backtracking method
//Input: n-number of queens
//Output: k-the queen k
// x[k]-the position of queen k
k←1
x[k]←0
while k!=0 **do**
x[k]←x[k]+1
while place(x,k)!=1 **and** x[k]<=n **do**
x[k]←x[k]+1
end while
if x[k]<=n
if k=n
for k←1 to n **do**
write k,x[k]
end for
else
k←k+1
x[k]←0
end if
else
k←k-1
end if
end while

ALGORITHM: place(x[k],k)
//places n queens on a nXn matrix such that no two queens are placed along same row or same column
//or same diagonal using backtracking method
//Input: k-the queen k
// x[k]- position of queen k
//Output: returns 0 if any two queens are placed along same diagonal or same column otherwise 1 is returned
for i←1 to k-1 **do**
if i-x[i]=k-x[k] **or** i+x[i]=k+x[k] **or** x[i]=x[k]
return 0
end if
end for

return 1

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void nqueens(int n)
{
    Int k,x[20],count=0;
    k=1;
    x[k]=0;
    while(k!=0)
    {
        x[k]++;
        while(place(x,k)!=1 && x[k]<=n)
            x[k]++;
        if(x[k]<=n)
        {
            if(k==n)
            {
                printf("\nSolution is %d\n", ++count);
                printf("Queen\t\tPosition\n");
                for(k=1;k<=n;k++)
                    printf("%d\t\t%d\n", k,x[k]);
            }
            else
            {
                k++;
                x[k]=0;
            }
        }
        else
            k--;
    }
}
int place(int x[], int k)
{
    int i;
    for(i=1;i<=k-1;i++)
    {
        if(i+x[i]==k+x[k]||i-x[i]==k-x[k]||x[i]==x[k])
            return 0;
    }
    return 1;
}
void main()
{
    int n;
    clrscr();
    printf("Enter the number of Queens\n");
    scanf("%d", &n);
    nqueens(n);
    getch();
}
```

Output:

```
Enter the number of Queens
4
Solution is 1
```

Queen Position
1 2
2 4
3 1
4 3

Solution is 2
Queen Position
1 3
2 1
3 4
4 2

EXPERIMENT 10

AIM: Write program to find the Binomial Co-efficient using Dynamic Programming.

Note: In the record book students should
- Handwrite the Algorithm,
- Handwrite the Program
- Pasting of the printout of the Output or handwrite the Output

ALGORITHM: bincof(n,r)
//computes binomial co-efficient c(n,r) by dynamic programming
//Input: Non-negative integers *n* and *r* such that n>=r>=0
//Output: The value of c(n,r)
for i←0 to n **do**
  for j←0 to r **do**
    if j=0
      c[i,j]←1
    else if i=j
      c[i,j]←1
    else if i>j
      c[i,j]←c[i-1,j]+c[i-1,j-1]
    end if
  end for
end for
write 'optimal solution is',c[n,r]

Program:

```
#include<stdio.h>
#include<conio.h>
int bin(intn,int k)
{
        inti,j,c[10][10];
        for(i=0;i<=n;i++)
        {
                for(j=0;j<=k;j++)
                {
                        if(j==0 || i==j)
                                c[i][j]=1;
                        else
                                c[i][j]=c[i-1][j-1]+c[i-1][j];
                }
        }
        return c[n][k];
}

void main()
{
        intn,k;
        printf("Enter the value of n & k such that n>k\n");
```

```
        scanf("%d%d", &n,&k);
        printf("C(%d,%d)=%d\n",n,k,bin(n,k));
        getch();
}
```

Output:
Enter the value of n & k such that n>k
3
2
C(3,2)=3
Enter the value of n & k such that n>k
5
3
C(5,3)=10

EXPERIMENT 11

AIM: Implement O/I Knapsack problem using dynamic programming.
Note: In the record book students should
- Handwrite the Algorithm,
- Handwrite the Program
- Pasting of the printout of the Output or handwrite the Output

ALGORITHM : knapsack(w[1…n],p[1…n],n,m)

```
//To find the optimal solution for the Knapsack problem using dynamic programming
// Input: n-number of objects to be selected
//        m-maximum capacity of the Knapsack
//        An array w[1….n] contains weights of all objects
//        An array p[1….n] contains profits of all objects
// Output :A matrix  v[0….n,0….m] contains the optimal solution for the number of  objects selected with
//          specified remaining capacity
for i←0 to n do
   for j←0 to m do
      if i=0 or j=0
         v[i,j]=0
      else if j-w[i]<0
         v[i,j]=v[i-1,j]
      else
         v[i,j]=max(v[i-1,j],v[i-1,j-w[i]+p[i])
      end if
   end for
end for
write 'the output is'
for i←0 to n do
   for j←0 to m do
      write v[i,j]
   end for
end for
write 'the optimal solution is',v[n,m]
write 'solution vector is'
for i←n downto 1 do
    if  v[i,m]!=v[i-1,m]
       x[i]←1
       m←m-w[i]
    else
       x[i]←0
    end if
end for
for i←1 to n do
```

write x[i]
end for
return

Program:

```
#include<stdio.h>
#include<conio.h>
void knapsack();
int max(int,int);
int i,j,n,m,p[10],w[10],v[10][10];
void main()
{
 clrscr();
 printf("\nenter the no. of items:\t");
 scanf("%d",&n);
 printf("\nenter the weight of the each item:\n");
 for(i=1;i<=n;i++)
 {
  scanf("%d",&w[i]);
 }
 printf("\nenter the profit of each item:\n");
 for(i=1;i<=n;i++)
 {
  scanf("%d",&p[i]);
 }
 printf("\nenter the knapsack's capacity:\t");
 scanf("%d",&m);
 knapsack();
 getch();
}

void knapsack()
{
 int x[10];
 for(i=0;i<=n;i++)
 {
  for(j=0;j<=m;j++)
  {
   if(i==0||j==0)
   {
    v[i][j]=0;
   }
   else if(j-w[i]<0)
   {
    v[i][j]=v[i-1][j];
   }
   else
   {
    v[i][j]=max(v[i-1][j],v[i-1][j-w[i]]+p[i]);
   }
  }
 }
 printf("\nthe output is:\n");
 for(i=0;i<=n;i++)
 {
  for(j=0;j<=m;j++)
  {
   printf("%d\t",v[i][j]);
```

```
 }
 printf("\n\n");
 }
 printf("\nthe optimal solution is %d",v[n][m]);
 printf("\nthe solution vector is:\n");
 for(i=n;i>=1;i--)
 {
  if(v[i][m]!=v[i-1][m])
  {
   x[i]=1;
   m=m-w[i];
  }
  else
  {
   x[i]=0;
  }
 }
 for(i=1;i<=n;i++)
 {
  printf("%d\t",x[i]);
 }
}

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

Output:

```
Enter the no. of items:  4

Enter the weight of each item:
2   1    3   2

Enter the profit of the each item:
12  10  20  15

Enter the Knapsack's capacity:  5

The output is:
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
The optimal solution is:   37

The solution vector is:
1   1   0   1
```

EXPERIMENT 12

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

Note: In the record book students should
- Handwrite the Algorithm,
- Handwrite the Program
- Pasting of the printout of the Output or handwrite the Output

ALGORITHM: floyds(a[1….n,1….n])
//Implements Floyd's algorithm for all-pairs shortest path problem
//Input: cost matrix a[1….n,1….n] of size nXn
//Output: Shortest distance matrix a[1….n,1….n] of size nXn
for k←1 to n **do**
  for i←1 to n **do**
    for j←1 to n **do**
      a[i,j]←min(a[i,j],a[i,k]+a[k,j])
    end for
  end for
end for
write 'all pair shortest path matrix is'
for i←1 to n **do**
  for j←1 to n **do**
    write a[i,j]
  end for
end for
Program:

```
#include<stdio.h>
#include<conio.h>
int a[10][10],n;
void floyds();
int min(int,int);
void main()
{
 int i,j;
 clrscr();
 printf("\nenter the no. of vertices:\t");
 scanf("%d",&n);
 printf("\nenter the cost matrix:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   scanf("%d",&a[i][j]);
  }
 }
 floyds();
 getch();
}

void floyds()
{
 int i,j,k;
 for(k=1;k<=n;k++)
 {
  for(i=1;i<=n;i++)
  {
   for(j=1;j<=n;j++)
   {
    a[i][j]=min(a[i][j],a[i][k]+a[k][j]);
   }
  }
```

```
 }
 printf("\nall pair shortest path matrix is:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   printf("%d\t",a[i][j]);
  }
  printf("\n\n");
 }
}

int min(int x,int y)
{
 if(x<y)
 {
  return x;
 }
 else
 {
  return y;
 }
}
```

Output

```
Enter the no. of vertices:  4

Enter the cost matrix:
9999   9999       3   9999
    2   9999   9999   9999
9999       7   9999       1
    6   9999   9999   9999

All pair shortest path matrix is:
10    10     3     4
 2    12     5     6
 7     7    10     1
 6    16     9    10
```

EXPERIMENT 13

AIM: Sort a given set of N integer elements using **Heap Sort** technique and compute its time taken. Run the program for different values of N and record the time taken to sort. Plot a graph of the time taken versus N using MS Excel. The program should allow both manual entry of the array elements and also reading of array elements using random number generator.

Note: In the record book students should

- Handwrite the Algorithm,
- Handwrite the Program
- Pasting of the printout of the Output and Graph or handwrite the Output and Graph.

ALGORITHM : build_heap(a[0….n-1])

// constructs a max heap from the elements in the given array

// Input : An array a[0….n-1] of orderable elements

//Output : a[0….n-1] contains a max heap

for i←(n-1)/2 downto 0 **do**

    heapify(a,n,p)

end for

ALGORITHM : heapify(a[0….n-1],p)

//create a heap for a subtree whose root node is identified as parent node ***p***

//Input : An array a[0….n-1] of orderable elements

//Output : The subtree whose root node was identified as parent node ***p*** will be in a heap

item←a[p]

```
c←2*p+1
while c<=n-1 do
   if c+1<=n-1
     if a[c]<a[c+1]
        c←c+1
     end if
   end if
   if item<a[c]
      a[p]←a[c]
      p←c
     c←2*p+1
  else
     break
  end if
end while
a[p]←item
ALGORITHM : heap_sort(a[0….n-1])
// To sort the items by using  heap
//Input : The items of array a[0….n-1] to be sorted
//Output : a[0…n-1] contains sorted items
for i←n-1 downto 0 do
    swap a[0] and a[i]
    build_heap(a,i)
end for
```

Program: Will be provided shortly

EXPERIMENT 14

AIM: Find Minimum Cost Spanning Tree of a given undirected graph using **Prim's algorithm.**

Note: In the record book students should
- Handwrite the Algorithm,
- Handwrite the Program
- Pasting of the printout of the Output or handwrite the Output

```
ALGORITHM: prims(c[1…n,1…n])
//To compute the minimum spanning tree of a given weighted undirected graph using Prim's
// algorithm
//Input: An nXn cost matrix c[1…n,1….n]
//Output: minimum cost of spanning tree of given undirected graph
ne←0
mincost←0
for i←1 to n do
    elec[i]←1
end for
elec[1]←1
while ne!=n-1 do
    min←9999
    for i←1 to n do
       for j←1 to n do
         if elec[i]=1
           if c[i,j]<min
              min←c[i,j]
              u←i
               v←j
          end if
         end if
       end for
   end for
```

```
    if elec[v]!=1
      write u,v,min
      elec[v]←1
      ne←ne+1
      mincost←mincost+min
    end if
    c[u,v]←9999
    c[v,u]←9999
end while
write mincost
return
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void prims();
int c[10][10],n;
void main()
{
 int i,j;
 clrscr();
 printf("\nenter the no. of vertices:\t");
 scanf("%d",&n);
 printf("\nenter the cost matrix:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   scanf("%d",&c[i][j]);
  }
 }
 prims();
 getch();
}

void prims()
{
 int i,j,u,v,min;
 int ne=0,mincost=0;
 int elec[10];
 for(i=1;i<=n;i++)
 {
  elec[i]=0;
 }
 elec[1]=1;
 while(ne!=n-1)
 {
  min=9999;
  for(i=1;i<=n;i++)
  {
   for(j=1;j<=n;j++)
   {
    if(elec[i]==1)
    {
     if(c[i][j]<min)
     {
      min=c[i][j];
```

```
      u=i;
      v=j;
     }
    }
   }
  }
 if(elec[v]!=1)
 {
  printf("\n%d----->%d=%d\n",u,v,min);
  elec[v]=1;
  ne=ne+1;
  mincost=mincost+min;
 }
 c[u][v]=c[v][u]=9999;
}
 printf("\nmincost=%d",mincost);
}
```

==========Output==========

```
Enter the no. of vertices:   6

Enter the cost matrix:
9999       3    9999    9999       6       5
    3    9999       1    9999    9999       4
9999       1    9999       6    9999       4
9999       6       6    9999       8       5
    6    9999    9999       8    9999       2
    5       4       4       5       2    9999

2----------> 3 = 1
5----------> 6 = 2
1----------> 2 = 3
2----------> 6 = 4
4----------> 6 = 5
Mincost = 15
```

EXPERIMENT 15

AIM: Find Minimum Cost Spanning Tree of a given undirected graph using **Kruskals algorithm**.

Note: In the record book students should
- Handwrite the Algorithm
- Handwrite the Program
- Pasting of the printout of the Output or handwrite the Output

ALGORITHM: kruskals(c[1…n,1…n])

```
//To compute the minimum spanning tree of a given weighted undirected graph using Kruskal's
// algorithm
//Input: An nXn cost matrix c[1…n,1….n]
//Output: minimum cost of spanning tree of given undirected graph
ne←0
mincost←0
for i←1 to n do
    parent[i]←0
end for
while ne!=n-1 do
    min←9999
    for i←1 to n do
        for j←1 to n do
            if c[i,j]<min
                min←c[i,j]
```

```
            u←i
            a←i
          v←j
          b←j
        end if
      end for
    end for
    while parent[u]!=0 do
        u←parent[u]
    end while
    while parent[v]!=0 do
        v←parent[v]
    end while
    if u!= v
      write a,b,min
      parent[v]←u
      ne←ne+1
      mincost←mincost+min
    end if
    c[a,b]←9999
    c[b,a]←9999
end while
write mincost
return
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void kruskals();
int c[10][10],n;
void main()
{
 int i,j;
 clrscr();
 printf("\nenter the no. of vertices:\t");
 scanf("%d",&n);
 printf("\nenter the cost matrix:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   scanf("%d",&c[i][j]);
  }
 }
 kruskals();
 getch();
}

void kruskals()
{
 int i,j,u,v,a,b,min;
 int ne=0,mincost=0;
 int parent[10];
 for(i=1;i<=n;i++)
 {
  parent[i]=0;
 }
 while(ne!=n-1)
```

```
{
 min=9999;
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
    if(c[i][j]<min)
    {
     min=c[i][j];
     u=a=i;
     v=b=j;
    }
  }
 }
 while(parent[u]!=0)
 {
  u=parent[u];
 }
 while(parent[v]!=0)
 {
  v=parent[v];
 }
 if(u!=v)
 {
  printf("\n%d----->%d=%d\n",a,b,min);
  parent[v]=u;
  ne=ne+1;
  mincost=mincost+min;
 }
 c[a][b]=c[b][a]=9999;
 }
 printf("\nmincost=%d",mincost);
}
```

==========Output==========

```
Enter the no. of vertices:   6

Enter the cost matrix:
9999       3    9999    9999       6       5
    3    9999       1    9999    9999       4
9999       1    9999       6    9999       4
9999       6       6    9999       8       5
    6    9999    9999       8    9999       2
    5       4       4       5       2    9999

2----------> 3 = 1
5----------> 6 = 2
1----------> 2 = 3
2----------> 6 = 4
4----------> 6 = 5
Mincost = 15
```

EXPERIMENT 16

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

Note: In the record book students should

- Handwrite the Algorithm
- Handwrite the Program
- Pasting of the printout of the Output or handwrite the Output

ALGORITHM: dijkstras(c[1….n,1….n],src)

//To compute shortest distance from given source node to all nodes of a weighted undirected graph

```
//Input: An nXn cost matrix c[1…n,1….n] with source node src
//Output: The length dist[j] of a shortest path from src to j
for j←1 to n do
    dist[j]←c[src,[j]
end for
for j←1 to n do
     vis[j]←0
end for
dist[src]←0
vis[src]←1
count←1
while count!=n do
       min←9999
       for j←1 to n do
          if dist[j]<min and vis[j]!=1
             min←dist[j]
             u←j
          end if
        end for
        vis[u]←1
        count←count+1
        for j←1 to n do
           if min+c[u,j]<dist[j] and vis[j]!=1
              dist[j]←min+c[u,j]
           end if
        end for
end while
write ‘shortest distance is’
for j←1 to n do
    write src,j,dist[j]
end for
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void dijkstras();
int c[10][10],n,src;
void main()
{
 int i,j;
 clrscr();
 printf("\nenter the no of vertices:\t");
 scanf("%d",&n);
 printf("\nenter the cost matrix:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   scanf("%d",&c[i][j]);
  }
 }
 printf("\nenter the source node:\t");
 scanf("%d",&src);
 dijkstras();
 getch();
}

void dijkstras()
```

```
{
 int vis[10],dist[10],u,j,count,min;
 for(j=1;j<=n;j++)
 {
  dist[j]=c[src][j];
 }
 for(j=1;j<=n;j++)
 {
  vis[j]=0;
 }
 dist[src]=0;
 vis[src]=1;
 count=1;
 while(count!=n)
 {
  min=9999;
  for(j=1;j<=n;j++)
  {
   if(dist[j]<min&&vis[j]!=1)
   {
    min=dist[j];
    u=j;
   }
  }
  vis[u]=1;
  count++;
  for(j=1;j<=n;j++)
  {
   if(min+c[u][j]<dist[j]&&vis[j]!=1)
   {
    dist[j]=min+c[u][j];
   }
  }
 }
 printf("\nthe shortest distance is:\n");
 for(j=1;j<=n;j++)
 {
  printf("\n%d----->%d=%d",src,j,dist[j]);
 }
}
```

==========Output==========

```
Enter the no. of vertices:   5

Enter the cost matrix:
9999       3   9999       7    9999
   3    9999       4       2    9999
9999       4   9999       5       6
   7       2       5    9999       4
9999    9999       6       4    9999

Enter the source node:   1

The shortest distance is:
1-----------> 1 = 0
1-----------> 2 = 3
1-----------> 3 = 7
1-----------> 4 = 5
1-----------> 5 = 9
```

PROGRAM OUTCOMES

PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO 6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO 7	Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO 8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO 9	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO 10	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO 11	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO 12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

CO-PO-PSO MAPPING

Course Code: 15CS5DCADA	**Course Title:** Analysis and Design of Algorithms													
Course Outcomes	**PO 1**	**PO 2**	**PO3**	**PO 4**	**PO5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**	**PSO 1**	**PSO 2**	**PSO3**
CO 1: Ability to analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.		3													
CO2: Ability to design efficient algorithms using various design techniques.			3												3
CO3: Ability to apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete	3														
CO4: Ability to conduct practical experiments to solve problems using an appropriate designing method and find time efficiency.				3											