PRESIDENCY UNIVERSITY

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

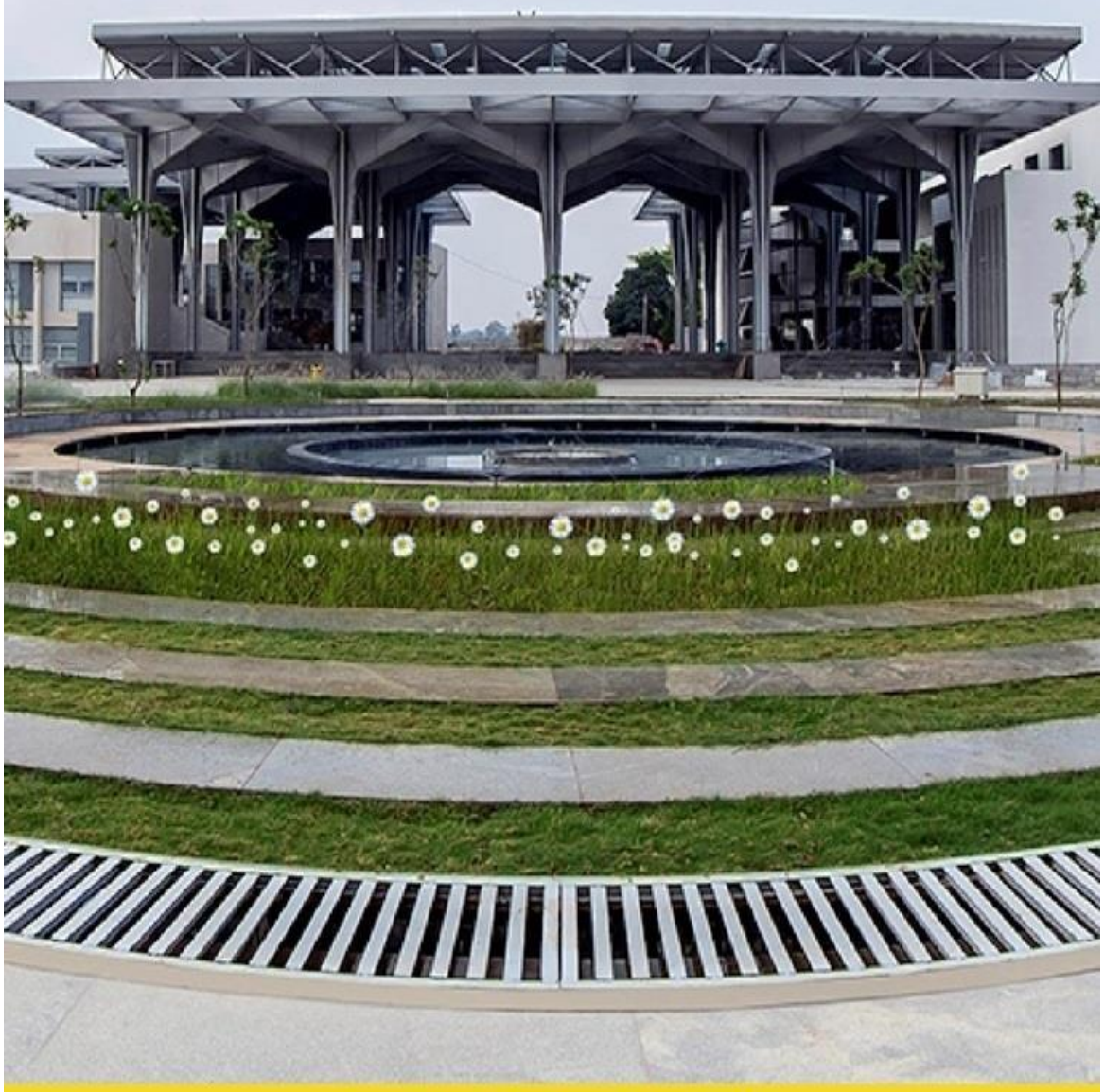
Private University Estd. in Karnataka State by Act No. 41 of 2013

PRESIDENCY GROUP
OVER 40 YEARS OF ACADEMIC WISDOM

Department Of Computer Science

CSE255: Analysis of Algorithms Lab

5th Semester 2018-19



www.presidencyuniversity.in

PRESIDENCY UNIVERISTY, BENGALURU
School of Engineering

PROGRAMME – B.Tech Computer Science and Engineering

V Semester 2018-19

ANALYSIS OF ALGORITHMS LAB (CSE255)

Course Handout

Course Code: CSE 255

Course Name: Analysis of Algorithms Lab

Credit Structure: 0-0-2-1

Instructor-in-charge : Mrutyunjaya M S

Instructor(s): Sunil Kumar R M, Deepak Raj, Shreekant Jere, Manjula H M,

Napa Lakshmi, Deepak Kumar, Mrutyunjaya M S, Vijaya Narasimha Murthy.

Programme Outcomes (Outcomes applicable to this course are in bold and their alphabets are circled)

On successful completion of the Programme the student shall have:

(a) an ability to apply knowledge of mathematics, science, and engineering

(b) an ability to design and conduct experiments, as well as to analyze and interpret data

c) an ability to design a system, component, or process to meet desired needs within realistic constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability

d) an ability to function on multidisciplinary teams

(e) an ability to identify, formulate, and solve engineering problems

f) an understanding of professional and ethical responsibility

g) an ability to communicate effectively

h) an ability to understand the impact of engineering solutions in a global, economic, environmental, and societal context

i) An ability to recognize the need for, and an ability to engage in life-long learning

j) an ability to acquire the knowledge of contemporary issues

k) an ability to use the techniques, skills, and modern engineering tools necessary for engineering practice

1. Learning Objectives and Outcomes of the Course:

Learning Objectives:

On successful completion of the course the students is able to:

LO1: Illustrate various types of algorithms.

LO2: Analyze time and space efficiency of Recursive/Non Recursive, Dynamic and greedy algorithms.

LO3: Identify various alternative solutions to a problem.

Outcomes of the Course:

On successful completion of the course the students is able to:

CO1: Analyze worst-case running times of algorithms using asymptotic notations.

CO2: Design and Code Divide & Conquer Algorithms

CO3: Design and Code Dynamic Programming Algorithms.

CO4: Design and Code Greedy Algorithms.

CO5: Design and Code Back Tracking Algorithms

2. Course Description:

This Course introduces techniques for the design and analysis of efficient algorithms and methods of applications. It deals with analyzing time and space complexity of algorithms, and to evaluate trade-offs between different algorithms.

Topics include: Brute force- Bubble sort, Linear search, Divide-and-conquer- Merge sort, Quick sort, Binary search. Dynamic programming and greedy technique- Prim’s, Kruskal's, Dijkstra’s Algorithm, Warshall’s algorithm, Floy’d algorithm, Backtracking – N Queens Problem.

3. Syllabus :

Non-recursive algorithms: Factorial, Max, Unique, Count, Matrix multiplication.

Recursive algorithms: Factorial, base conversion, GCD, Search, Tower of Hanoi.

Brute Force Technique: Bubble sort, Linear Search.

Divide and Conquer: Binary Search, merge sort, quick sort.

The Greedy Method: Prim's and Kruskal's algorithm to find Minimum Spanning Tree, Single source shortest path (Djikstra's Algorithm).

Dynamic programming: The knapsack problem, Warshall's Algorithm, Floyd's Algorithm.

Backtracking: N-Queens problem.

4. Instructional Pedagogy:

This course will enable students to learn to implement various algorithms. The method of learning is by practicing in the lab. Students will be oriented towards the concept using suitable examples. At times, presentations and case studies will be used to add to the learning.

5. Books:

(i) Textbook(s)

1. Thomas H.Cormen, Charles E.Leiserson, Ronald L. Rivest and Clifford Stein, "Introduction to Algorithms", PHI Learning Private Limited.

(ii) **Reference Book(s)**

1. Anany Levitin, "Introduction to the Design and Analysis of Algorithms", Pearson Education.

2. Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman, "Data Structures and Algorithms", Pearson.

6. **SESSION PLAN**

Lab No.	**Learning Objective(s)**	**Assignment to be completed**
1	Explain non recursive algorithms, calculating time and space efficiency (best, average & worst)	Implement Factorial of a number, finding max element, linear search and find their time and space efficiency.
2	Explain non recursive algorithms, calculating time and space efficiency (best, average & worst)	Matrix multiplication, bubble sort and find their time and space efficiency.
3	Explain recursive algorithms, calculating time and space efficiency (best, average & worst)	Implement Factorial of a number, finding GCD using Euclid's method, tower of Hanoi and find their time and space efficiency.
4	Explain sorting techniques, calculating time and space efficiency (best, average & worst)	Implement, Merge sort, quick sort and compare their time and space tradeoff.
5	Explain concept of dynamic programming and find time and space efficiency (best, average & worst)	Implement nCr, find space and time tradeoff for different inputs.
6	Solving 0/1 knapsack problem and finding its efficiency.	Implement knapsack algorithm for variety of inputs and find its efficiency
7	Explain single source shortest path problem – Dijkstra's algorithm	Implement Dijkstra's algorithm for different graphs and find its efficiency.
8	Explain concept of constructing minimum spanning tree using Kruskal's algorithm.	Implement Kruskal's algorithm for different graphs, find its efficiency and construct MST

Lab No.	Learning Objective(s)	Assignment to be completed
9	Explain constructing MST using prim's algorithm	Implement Prim's algorithm for different graphs and find its efficiency and construct MST.
10	Explain travelling sales person problem	Implement TSP for different inputs, find its efficiency and construct best route.
11	Explain All pair Shortest Path problem	Implement Floyds and Warshall's algorithm for different graphs.
12	Explain N – Queens problem	Implement queen's problems for 4, 8 and 16 inputs.
13	Review session	Doubt clearing, Review of the key concepts of algorithm analysis

7. Assessments:

Component	Duration (minutes)	% Weightage	Marks	Date & Time	Venue
Laboratory Exercise conducted during periodic lab sessions/Record/ Continuous assessment	-	40%	80	During Lab hours	Lab
Mid-term Lab Exam	120 Minutes	20%	40	TBD	Lab
Assignments and/or Projects *	Take home	10%	20	To be announced in the	-

				class	
Comprehensive Lab Exam	120 Minutes	30%	60	TBD	Lab

* Details to be announced in the class and course website

8. Chamber consultation hour: Will be announced in the class.

9. Notices: All notices related to the course will be displayed on the Notice Board and uploaded on Course website.

10. Note:

1. No Make-up will be available for any of the assignments.

2. A make-up test shall be granted only in genuine cases where - in the Instructor In-Charge's judgment - the student would be physically unable to appear for the test.

3. Assignments/projects will be evaluated through a Viva-Voce.

4. Details of assignments/projects including problem specifications, deliverables, team sizes, deadlines, and Viva-voce schedule will be posted on the course website

Session 1

Non recursive algorithms

a. Implement Factorial of a number

b. finding max element

c. linear search

d. find their time and space efficiency.

Algorithm

ALGORITHM Factorial of a number *(n)*
Step 1: Start
Step 2: Declare variables n, factorial and i.
Step 3: Initialize variables
factorial←1
i←1
Step 4: Read value of n
Step 5: Repeat the steps until i=n
5.1: factorial←factorial*i
5.2: i←i+1
Step 6: Display factorial
Step 7: Stop

ALGORITHM finding max element
SET Max to array[0]
FOR i = 1 to array length - 1
IF array[i] > Max THEN
SET Max to array[i]
ENDIF
ENDFOR
PRINT Max

ALGORITHM linear search/sequential search
FOR i = 0 to array length - 1
IF X = array[i] THEN
RETURN i
ENDIF
ENDFOR
RETURN -1

Coding using C Language

a. Implement Factorial of a number.

```
#include <stdio.h>
int main()
{
   int n, i;
   unsigned long long factorial = 1;
   printf("Enter an integer: ");
   scanf("%d",&n);
// show error if the user enters a negative integer
   if (n < 0)
      printf("Error! Factorial of a negative number doesn't exist.");
 else
   {
      for(i=1; i<=n; ++i)
      {
         factorial *= i;              // factorial = factorial*i;
      }
      printf("Factorial of %d = %llu", n, factorial);
   }
return 0;
}
```

b. Finding max element

```
#include <stdio.h>
int main()
{
   int i, n;
   float arr[100];
   printf("Enter total number of elements(1 to 100): ");
   scanf("%d", &n);
   printf("\n");
   // Stores number entered by the user
   for(i = 0; i < n; ++i)
   {
     printf("Enter Number %d: ", i+1);
     scanf("%f", &arr[i]);
   }
   // Loop to store largest number to arr[0]
   for(i = 1; i < n; ++i)
   {
     // Change < to > if you want to find the smallest element
     if(arr[0] < arr[i])
        arr[0] = arr[i];
   }
   printf("Largest element = %.2f", arr[0]);
return 0;
}
```

c. linear search/sequential search

```
#include<stdio.h>
 main()
 {
   int array[100], search, c, number;
   printf("Enter the number of elements in array\n");
   scanf("%d",&number);
   printf("Enter %d numbers\n", number);
   for ( c = 0 ; c < number ; c++ )
     scanf("%d",&array[c]);
   printf("Enter the number to search\n");
   scanf("%d",&search);
   for ( c = 0 ; c < number ; c++ )
   {
     if ( array[c] == search )   /* if required element found */
     {
       printf("%d is present at location %d.\n", search, c+1);
       break;
     }
   }
   if ( c == number )
     printf("%d is not present in array.\n", search);
   return 0;
 }
```

Session 2

a. Matrix multiplication
b. bubble sort

ALGORITHM matrix multiplication.

1. Start
2. Declare variables and initialize necessary variables
3. Enter the element of matrices by row wise using loops
4. Check the number of rows and column of first and second matrices
5. If number of rows of first matrix is equal to the number of columns of second matrix, go to step 6. Otherwise, print matrix multiplication is not possible and go to step 3.
6. Multiply the matrices using nested loops.
7. Print the product in matrix form as console output.
8. Stop

ALGORITHM BubbleSort(list)

```
begin BubbleSort(list)
for all elements of list
     if list[i] > list[i+1]
        swap(list[i], list[i+1])
     end if
  end for
  return list
end BubbleSort
```

Coding using C Language

a. Matrix multiplication

```
#include <stdio.h>
int main()
{
 int m, n, p, q, c, d, k, sum = 0;
 int first[10][10], second[10][10], multiply[10][10];
 printf("Enter number of rows and columns of first matrix\n");
 scanf("%d%d", &m, &n);
 printf("Enter elements of first matrix\n");
 for (c = 0; c < m; c++)
   for (d = 0; d < n; d++)
     scanf("%d", &first[c][d]);
 printf("Enter number of rows and columns of second matrix\n");
 scanf("%d%d", &p, &q);
 if (n != p)
   printf("The matrices can't be multiplied with each other.\n");
 else
```

```
  {
    printf("Enter elements of second matrix\n");
    for (c = 0; c < p; c++)
      for (d = 0; d < q; d++)
        scanf("%d", &second[c][d]);
    for (c = 0; c < m; c++) {
      for (d = 0; d < q; d++) {
        for (k = 0; k < p; k++) {
          sum = sum + first[c][k]*second[k][d];
        }
   multiply[c][d] = sum;
        sum = 0;
      }
    }
    printf("Product of the matrices:\n");
    for (c = 0; c < m; c++) {
      for (d = 0; d < q; d++)
        printf("%d\t", multiply[c][d]);
      printf("\n");
    }
  }
  return 0;
}
```

b. Bubble Sort

```
#include <stdio.h>
int main()
{
int data[100],i,n,step,temp;
    printf("Enter the number of elements to be sorted: ");
    scanf("%d",&n);
    for(i=0;i<n;++i)
  {
        printf("%d. Enter element: ",i+1);
        scanf("%d",&data[i]);
    }
    for(step=0;step<n-1;++step)
    for(i=0;i<n-step-1;++i)
    {
        if(data[i]>data[i+1])
        {
            temp=data[i];
            data[i]=data[i+1];
            data[i+1]=temp;
        }
    }
printf("In ascending order: ");
    for(i=0;i<n;++i)
         printf("%d  ",data[i]);
    return 0;}
```

Session 3

Recursive algorithms

a. Factorial of a number

b. finding GCD using Euclid's method

c. tower of Hanoi

Algorithms

ALGORITHM Factorial of a number(n)
Step 1: Start
Step 2: Read number n
Step 3: : If n==1 then return 1
Step 4: Else
f=n*factorial(n-1)
Step 4: Print factorial f
Step 5: Stop

ALGORITHM euclids(m,n)
function gcd(a, b)
if b = 0
return a;
else
return gcd(b, a mod b);

ALGORITHM Towerofhanoi
START
Procedure Hanoi(disk, source, dest, aux)
IF disk == 1, THEN
move disk from source to dest
ELSE
Hanoi(disk - 1, source, aux, dest) // Step 1
move disk from source to dest // Step 2
Hanoi(disk - 1, aux, dest, source) // Step 3
END IF
END Procedure
STOP

Coding using C Language

a. Factorial of a number

```
#include <stdio.h>
 int factorial(int);
 int main()
{
   int num;
```

```
    int result;
     printf("Enter a number to find it's Factorial: ");
    scanf("%d", &num);
    if (num < 0)
    {
       printf("Factorial of negative number not possible\n");
    }
    else
    {
       result = factorial(num);
       printf("The Factorial of %d is %d.\n", num, result);
    }
    return 0;
}
int factorial(int num)
{
    if (num == 0 || num == 1)
    {
       return 1;
    }
    else
    {
       return(num * factorial(num - 1));
    }
}
```

b. Find gcd of two numbers using euclids algorithm

```
#include <stdio.h>
int gcd(int a, int b)
{
   if (b == 0)
      return a;
   return gcd(b, a % b);
}
int main()
{
   int a = 98, b = 56;
   printf("GCD of %d and %d is %d ", a, b, gcd(a, b));
   return 0;
}
```

c. Tower of Hanoi

```
#include <stdio.h>
// C recursive function to solve tower of hanoi puzzle
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
{
   if (n == 1)
   {
      printf("\n Move disk 1 from rod %c to rod %c", from_rod, to_rod);
```

```
        return;
    }
    towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
    printf("\n Move disk %d from rod %c to rod %c", n, from_rod, to_rod);
    towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
}
  int main()
{
    int n = 4; // Number of disks
    towerOfHanoi(n, 'A', 'C', 'B');  // A, B and C are names of rods
    return 0;
}
```

Session 4

Divide and Conquer Technique

a. Search for a given key in a set of elements using Binary Search algorithm

b. Sort a set of elements in ascending order using Merge Sort algorithm.

c. Sort a set of elements in ascending order using Quick Sort algorithm.

Algorithm

a. ALGORITHM BinSearchl(x, n, a[])

```
{
low := 1; high := n + 1; // high is one more than possible.
while (low < (high — 1)) do
{

mid := [(low + high)/2};
if (x < a[mid]) then high := mid; // Only one comparison in the loop.
else low := mid; // x > a[mid]
}
if (x = a[low]) then return low; // x is present.
else return 0;
}
```

b. ALGORITHM mergesort

Step 1 − if it is only one element in the list it is already sorted, return.
Step 2 − divide the list recursively into two halves until it can no more be divided.
Step 3 − merge the smaller lists into new list in sorted order.

c. ALGORITHM quicksort

Step 1 − Make the right-most index value pivot
Step 2 − partition the array using pivot value
Step 3 − quicksort left partition recursively
Step 4 − quicksort right partition recursively

Coding using C Language

a. Binary search

```
#include<stdio.h>
int main()
{
int a[10];
int key;
printf("enter number of the elements \n");
scanf("%d",&n);
printf("read n elements")
for(int i=0;i<n;i++)
scanf("%d",&a[i]);
printf("enter element to be searched")
scanf("%d",&key);
flag=bs(a,0,n-1,key);
if(flag==-1)
{
printf("element not found\n");
}
else
{
printf("element found at %d",flag+1);
}
}
bs(int a[],int low,int high,int key)
{
int mid;
if(low<=high)
{
mid=(low+high)/2;
if(a[mid]==key)
return mid;
if(a[mid]>=key)
bs(a,0,mid-1);
else
bs(a,mid+1,high);
}
return -1;
}
```

b. Merge sort

```
#include<stdio.h>
#define MAX 50
void mergeSort(int arr[],int low,int mid,int high);
void partition(int arr[],int low,int high);
int main()
{
int merge[MAX],i,n;
printf("Enter the total number of elements: ");
scanf("%d",&n);
printf("Enter the elements which to be sort:");
```

```
for(i=0;i<n;i++)
{
 scanf("%d",&merge[i]);
}
   partition(merge,0,n-1);
   printf("After merge sorting elements are: ");
    for(i=0;i<n;i++){
        printf("%d ",merge[i]);
    }
   return 0;
}
void partition(int arr[],int low,int high)
{
    int mid;
    if(low<high){
        mid=(low+high)/2;
        partition(arr,low,mid);
        partition(arr,mid+1,high);
        mergeSort(arr,low,mid,high);
    }
}
void mergeSort(int arr[],int low,int mid,int high){
    int i,m,k,l,temp[MAX];
    l=low;
    i=low;
    m=mid+1;
    while((l<=mid)&&(m<=high)){
        if(arr[l]<=arr[m]){
            temp[i]=arr[l];
            l++;}
        else{
            temp[i]=arr[m];
            m++;}
        i++;
    }
    if(l>mid){
        for(k=m;k<=high;k++){
            temp[i]=arr[k];
            i++;}
    }
    else{
        for(k=l;k<=mid;k++){
            temp[i]=arr[k];
            i++;
        }
    }
for(k=low;k<=high;k++){
        arr[k]=temp[k];
    }
}
```

c. Quick sort

```
#include<stdio.h>
# include<math.h>
#define max 500
void qsort(int [],int,int);
int partition(int [],int,int);
int main()
{
	int a[max],i,n;
	printf("Enter the value of n:\n");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	scanf("%d",&a[i]);
	printf("\nThe array elements before\n");
	for(i=0;i<n;i++)
		printf("%d\t",a[i]);
	qsort(a,0,n-1);
	printf("\nElements of the array after sorting are:\n");
	for(i=0;i<n;i++)
	printf("%d\t",a[i]);
}
void qsort(int a[],int low,int high)
{
int j;
if(low<high)
	{
		j=partition(a,low,high);
		qsort(a,low,j-1);
		qsort(a,j+1,high);
	}
}

int partition(int a[], int low,int high)
{
int pivot,i,j,temp;
pivot=a[low];
i=low+1;
j=high;
while(1)
{
	while(pivot>a[i] && i<=high)
	i++;
	while(pivot<a[j])
	j--;
	if(i<j)
	{
		temp=a[i];
		a[i]=a[j];
		a[j]=temp;
	}
```

```
        else
        {
                temp=a[j];
                a[j]=a[low];
                a[low]=temp;
                return j;
        }
}
}
```

Session 5

Dynamic programming

a. Binomial coefficient(nCr)
b. Warshal's algorithm to find transitive closure
c. Floyds algorithm to solve all pair shortest path problem
d. knapsack

Algorithms

Algorithm Binomial(n, k)
for i ← 0 to n do // fill out the table row wise
for i = 0 to min(i, k) do
if j==0 or j==i then C[i, j] ← 1 // IC
else C[i, j] ← C[i-1, j-1] + C[i-1, j] // recursive relation
return C[n, k]

Algorithm Warshal's

Input: Adjacency matrix A of relation R on a set of n elements
Output: Adjacency matrix T of the transitive closure of R.
T := A [initialize T to A]
for j := 1 to n
for i := 1 to n
if Ti, j = 1 then
i a := i j a ∨ a [form the Boolean OR of row i and row j, store it in i a]
next i
next j
end

```
ALGORITHM Floyd(W[1..n, 1..n])
    //Implements Floyd's algorithm for the all-pairs shortest-paths problem
    //Input: The weight matrix W of a graph with no negative-length cycle
    //Output: The distance matrix of the shortest paths' lengths
    D ← W //is not necessary if W can be overwritten
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                D[i, j] ← min{D[i, j], D[i, k] + D[k, j]}
    return D
```

Algorithm DPKnapsack(w[1..n], v[1..n], W)

```
var V[0..n,0..W], P[1..n,1..W];
for j := 0 to W do
V[0,j] := 0
for i := 0 to n do
V[i,0] := 0
for i := 1 to n do
for j := 1 to W do
if w[i] <= j and v[i] + V[i-1,j-w[i]] > V[i-1,j] then V[i,j] := v[i] + V[i-1,j-w[i]];
P[i,j] := j-w[i]
else V[i,j] := V[i-1,j]; P[i,j] := j
return V[n,W] and the optimal subset by backtracing
```

Coding using C Language

Binomial coefficient (nCr)

```
#include<stdio.h>
int min(int a, int b);
int binomialCoeff(int n, int k)
{
  int C[n+1][k+1];
    int i, j;
    for (i = 0; i <= n; i++)
    {
        for (j = 0; j <= min(i, k); j++)
        {
            if (j == 0 || j == i)
                C[i][j] = 1;
            else
                C[i][j] = C[i-1][j-1] + C[i-1][j];
        }
    }
  return C[n][k];
}
  int min(int a, int b)
{
    return (a<b)? a: b;
```

```
}

int main()
{       int n,k;
        printf("enter the values of n and k");
        scanf("%d%d",&n,&k);
printf ("Value of C(%d, %d) is %d ", n, k,binomialCoeff(n, k) );
        return 0;
}
```

Warshal's Algorithm

```
#include<stdio.h>
int max(int,int);
void warshal(int p[10][10],int n)
{
 int i,j,k;
 for(k=1;k<=n;k++)
  for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
    p[i][j]=max(p[i][j],p[i][k] && p[k][j]);
}
int max(int a,int b)
{
if(a>b)
 return(a);
else
 return(b);
}
void main()
{
 int p[10][10]={0},n,e,u,v,i,j;
 printf("\n Enter the number of vertices:");
 scanf("%d",&n);
 printf("\n Enter the number of edges:");
 scanf("%d",&e);
 for(i=1;i<=e;i++)
 {
  printf("\n Enter the end vertices of edge %d:",i);
  scanf("%d%d",&u,&v);
  p[u][v]=1;
 }
 printf("n Matrix of input data: n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
   printf("%d\t",p[i][j]);
  printf("\n");
 }
 warshal(p,n);
```

```
 printf("\n Transitive closure: \n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
   printf("%d\t",p[i][j]);
  printf("\n");
 }
}
```

Floyd's algorithm

```
#include<stdio.h>
int min(int,int);
void floyds(int p[10][10],int n)
{
 int i,j,k;
 for(k=1;k<=n;k++)
  for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
    if(i==j)
     p[i][j]=0;
    else
     p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
int min(int a,int b)
{
 if(a<b)
  return(a);
 else
  return(b);
}
void main()
{
 int p[10][10],w,n,e,u,v,i,j;
printf("\n Enter the number of vertices:");
 scanf("%d",&n);
 printf("\n Enter the number of edges:\n");
 scanf("%d",&e);
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
           p[i][j]=999;
 }
 for(i=1;i<=e;i++)
 {
  printf("\n Enter the end vertices of edge%d with its weight \n",i);
  scanf("%d%d%d",&u,&v,&w);
  p[u][v]=w;
 }
 printf("\n Matrix of input data:\n");
```

```
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
   printf("%d\t",p[i][j]);
  printf("\n");
 }
 floyds(p,n);
 printf("\n Transitive closure:\n");
 for(i=1;i<=n;i++)
 {
for(j=1;j<=n;j++)
   printf("%d\t",p[i][j]);
  printf("\n");
 }
 printf("\n The shortest paths are:\n");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  {
   if(i!=j)
    printf("\n <%d,%d>=%d",i,j,p[i][j]);
  }
}
```

KNAPSACK(0/1)

```
#include<stdio.h>
int v[20][20],values[23],weight[45];
int max(int m,int n)
{
        if(m>=n) return m;
        else return n;
}
int knapsack(int i,int j)
{
        int value;
        if(v[i][j]<0)
        {
                if(j-weight[i]>=0)
                {
                        value=max(knapsack(i-1,j),(values[i]+knapsack(i-1,j-weight[i])));
                }
                else
                {
                        value=knapsack(i-1,j);
                }
                v[i][j]=value;
        }
        return v[i][j];
}
main()
{
```

```
        int n,i,j,w,profit;
        printf("enter the no of elem\n");
        scanf("%d",&n);
        printf("enter the capacity\n");
        scanf("%d",&w);
        printf("enter the weights\n");
        for(i=1;i<=n;i++)
        {
                scanf("%d",&weight[i]);
        }
        printf("enter the profits\n");
        for(i=1;i<=n;i++)
        {
                scanf("%d",&values[i]);
        }
        for(i=0;i<=n;i++)
        for(j=0;j<=w;j++)
        if(i==0||j==0) v[i][j]=0;
        else v[i][j]=-1;
        profit=knapsack(n,w);
        printf("the value is %d\n",profit);

}
```

Session 6

Greedy Techniques
a. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm b. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm c. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijikstra's algorithm
Algorithms

ALGORITHM Kruskal(G) //Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph G = (V, E)
//Output: ET , the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights w(ei 1) ≤ . . . ≤ w(ei| E|)
ET ←NULL;
ecounter ←0 //initialize the set of tree edges and its size
k←0 //initialize the number of processed edges
while ecounter < | V| − 1 do
k← k + 1
if ET ∪ { eik} is acyclic
ET← ET ∪ { eik};
ecounter ← ecounter + 1
return ET.
 	
ALGORITHM Prim(G) //Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph *G* = _ *V*, *E* _
//Output: *ET* , the set of edges composing a minimum spanning tree of *G*

VT←{ *v* 0} //the set of tree vertices can be initialized with any vertex
ET←NULL
for *i* ←1 to | *V*| − 1 do
find a minimum-weight edge *e* = (v*, u*)* among all the edges *(v, u)*
such that *v* is in *VT* and *u* is in
V − *VT VT*← *VT* ∪ { *u**}
ET← *ET* ∪ { *e**}
return *ET*.

ALGORITHM Dijkstra's

Let's call the node we are starting with an **initial node**. Let a **distance of a node X** be the distance from the **initial node** to it. Our algorithm will assign some initial distance values and will try to improve them step-by-step.

1. Assign to every node a distance value. Set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes as unvisited. Set initial node as current.
3. For current node, consider all its unvisited neighbours and calculate their distance (from the initial node). For example, if current node (A) has distance of 6, and an edge connecting it with another node (B) is 2, the distance to B through A will be 6+2=8. If this distance is less than the previously recorded distance (infinity in the beginning, zero for the initial node), overwrite the distance.
4. When we are done considering all neighbours of the current node, mark it as visited. A visited node will not be checked ever again; its distance recorded now is final and minimal.
5. Set the unvisited node with the smallest distance (from the initial node) as the next "current node" and continue from step 3.
6. When all nodes are visited, algorithm ends.

Coding using C Language

a) Using Kruskal's Algorithm

```
#include<stdio.h>
int i,j,k,a,b,v,u,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
void main()
 {
   printf("\nEnter the number of vertices\n");
   scanf("%d",&n);
   printf("Enter the cost of matrix::\n");
   for (i=1;i<=n;i++)
         for (j=1;j<=n;j++)
         {
           scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                    cost[i][j]=999;
         }
         printf("\nThe edges of spanning treeare:\n\n");
   while(ne<n)
           {
             for (i=1,min=999;i<=n;i++)
                   for (j=1;j<=n;j++)
                     {
```

```
                    if(cost[i][j]<min)
                      {
                      min=cost[i][j];
                              a=u=i;
                              b=v=j;
                       }}
                while(parent[u])  u=parent[u];
                while(parent[v])  v=parent[v];
                if(u!=v)
                     {
                      printf("\n%d\tEdge(%d,%d)=%d",ne++,a,b,min);
                      mincost+=min;
                      parent[v]=u;
                     }
                    cost[a][b]=cost[b][a]=999;
               }
                printf("\n\tMiINCOST=%d\n",mincost);

  }
```

b) USING PRIM'S ALGORITHM

```
#include<stdio.h>
int i, j, a, b, v, u,n, ne=1;
int  min,mincost=0, cost[9][9], visited[9];
main()
{
          printf( "The no of vertices=\t");
          scanf("%d",&n);
          printf("The cost of matrix=\t");
         for( i=1;i<=n;i++)
         for( j=1;j<=n;j++)
         {
         scanf("%d",&cost[i][j]);
          if(cost[i][j]==0) cost[i][j]=999;
          }

printf("The edges of spanning tree are \t");
`         visited[1]=1;
          while(ne<n)
            {
                  for(i=1,min=999;i<=n; i++)
                   {
                    for(j=1;j<=n;j++)
                    {
                     if(cost[i][j]<min)
                     {
                     if(visited[i]==0)
                      continue;
                     else
```

```
                    {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
                    }
                   }
                  }
                  }

             if(visited[v]==0)
            {
            printf("\n%d\t Edge \t(%d, %d)=%d\n",ne++, a, b, min);
                    mincost+=min;
                    visited[b]=1;
                    }
              cost[a][b]=cost[b][a]=999;
              }
            printf("\n\t mincost=%d\n",mincost);

}
```

DIJKSTRAS PROGRAM

```
#include<stdio.h>
main()
{
  int visited[23],i,k,w,v,min;
  int n,sv,j,dist[10],cost[10][10];
  printf("\nenter the no of vertices");
  scanf("%d",&n);
  printf("enter the cost of matrix:\n");
  for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
     {
      scanf("%d",&cost[i][j]);
      if(cost[i][j]==0)
           cost[i][j]=999;
    }
    printf("enter the source\n");
    scanf("%d",&sv);
    for(i=1;i<=n;i++)
    {
      visited[i]=0;
      dist[i]=cost[sv][i];
   }
   visited[sv]=1;
   dist[sv]=1;
   for(k=2;k<=n;k++)
   {
    min=999;
```

```
   for(w=1;w<=n;w++)
   if(dist[w]<min && visited[w]==0)
   {
     min=dist[w];
     v=w;
   }
   visited[v]=1;
   for(w=1;w<=n;w++)
   if(dist[v]+cost[v][w]<dist[w])
    dist[w]=cost[v][w]+dist[v];
 }
 printf("shortest path\n");
 for(j=1;j<=n;j++)
 {
  if(j!=sv)
  printf("%d->%d==%d\n",sv,j,dist[j]);
 }
}
```

Session 7

a. Implement N Queen's problem using Back Tracking.

Algorithm

AlGORITHM Backtrack(X[l..i])
//Gives a template of a generic backtracking algorithm
//Input: X[Li] specifies first i promising components of a solution
//Output: All the tuples representing the problem's solutions

if X[Li] is a solution write X[l..i]
else
for each element x E S;+J consistent with X[Li] and the constraints do
X[i + 1] *-X
Backtrack(X[1..i + 1]J

Coding using C Language

a. Implement N Queen's problem using Back Tracking.

```
#include<stdio.h>
#include<stdlib.h>
const  int  max=20;
int   place(int x[],int k)
{
       int i;
       for(i=0;i<k;i++)
                if (x[i]==x[k] || abs(x[i]-x[k])==abs(i-k))
                        return 0;
       return 1;

}


void display(int x[], int n)
{
         char chessb[max][max];
         int i,j;
         for(i=0;i<n;i++)
               for(j=0;j<n;j++)
                        chessb[i][j]='x';
         for(i=0;i<n;i++)
               chessb[i][x[i]]='q';

         for(i=0;i<n;i++)
        {
```

```
            for(j=0;j<n;j++)
                  printf("%c", chessb[i][j]);

             printf ("\n");
         }

        printf ("\n");

}


void     nqueens(int n)
{
       int x[max];
       int k;
       x[0]=-1;
       k=0;
       while(k>=0)
      {
            x[k]=x[k]+1;
            while(x[k]<n && !place(x,k))
                     x[k]=x[k]+1;
            if(x[k]<n)
                    if (k==n-1)
                          display(x,n);
                     else
                    {
                           k++;
                           x[k]=-1;
                    }
            else
                   k--;
      }
}

main()
{
       int n;
       printf("enter the no of queens\n");
       scanf("%d", &n);
       printf("the soln to %d queens problem is \n", n);
       nqueens(n);
 }
```