ATME COLLEGE OF ENGINEERING

13th KM Stone, Bannur Road, Mysore - 560028

A T M E
College of Engineering

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING(DATA SCIENCE)

(ACADEMIC YEAR 2024-25)

LABORATORY MANUAL

SUBJECT : ANALYSIS & DESIGN OF ALGORITHMS LABORATORY

SUB CODE: BCSL404

SEMESTER: IV-2022 CBCS Scheme

Composed by	Verified by	Approved by
	Mrs. MADHU NAGARAJ	Dr. ANITHA D B
	FACULTY COORDINATOR	HOD, CSE- DS

INSTITUTIONAL MISSION AND VISION

Objectives

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied researchin areas of social relevance, and develop the institute as a center of excellence.
- To develop academic, professional and financial alliances with the industry as well asthe academia at national and transnational levels
- To develop academic, professional and financial alliances with the industry as well asthe academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

Vision

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING AND ENGINEERING

(DATA SCIENCE &ENGINEERING)

Vision of the Department

- To impart technical education in the field of data science of excellent quality with a high level of professional competence, social responsibility, and global awareness among the students

Mission

- To impart technical education that is up to date, relevant and makes students competitive and employable at global level
- To provide technical education with a high sense of discipline, social relevance in an intellectually, ethically and socially challenging environment for better tomorrow
- Educate to the global standards with a benchmark of excellence and to kindle the spirit of innovation.

Program Outcomes(PO)

- **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

- **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.
- **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.

- **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.

- **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.

- **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
- **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.

- **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

- **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

- **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.
- **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.
- **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.

Program Specific Outcomes (PSOs)

PSO1: Develop relevant programming skills to become a successful data scientist

- PSO2: Apply data science concepts and algorithms to solve real world problems of the society
- PSO3: Apply data science techniques in the various domains like agriculture, education healthcare for better society

Program Educational Objectives (PEOs):

PEO1: Develop cutting-edge skills in data science and its related technologies, such as machine learning, predictive analytic, and data engineering.

PEO2: Design and develop data-driven solutions to real-world problems in a business, research, or social environment.

PEO3: Apply data engineering and data visualization techniques to discover, investigate, and interpret data.

PEO4: Demonstrate ethical and responsible data practices in problem solving

PEO5: Integrate fields within computer science, optimization, and statistics to develop better solutions

Sl.No	Experiments
1.	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.
2.	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.
3.	a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm
4.	Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.
5.	Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph.
6.	Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.
7.	Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.
8.	Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d.
9.	Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.
10.	Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator
11.	Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator

1. **Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.**

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

int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];

int find(int);
int uni(int,int);
void main()
{
printf("\n\tImplementation of Kruskal's algorithm\n");
printf("\nEnter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the cost adjacency 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("The edges of Minimum Cost Spanning Tree are\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;
}
}
}
u=find(u);
v=find(v);
if(uni(u,v))
{
printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
mincost +=min;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
}
int find(int i)
```

```
{
while(parent[i])
i=parent[i];
return i;
}
int uni(int i,int j)
{
if(i!=j)
{
parent[j]=i;
return 1;
}
return 0;
}
```

2. Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
#include<stdio.h>
#include<conio.h>
int i,j,u,v,n,ne=1;
int min,mincost=0,visited[10]={0},cost[10][10];
void main()
{

 printf("\n Enter the No of nodes or vertices:");
 scanf("%d",&n);
 printf("\n Enter the Cost Adjacency matrix of the given graph:\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("The edges of Minimum Cost Spanning Tree are\n");
   visited[1]=1;
   printf("\n");
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)
   {
      min=cost[i][j];
      u=i;
      v=j;
   }
if(visited[u]==0 || visited[v]==0)
{
  printf("%d edge (%d,%d) =%d\n",ne++,u,v,min);
   mincost=mincost+min;
```

```
    visited[v]=1;
    }
    cost[u][v]=cost[v][u]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
}
```

3. a) Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.

```
#include<stdio.h>
#include<stdlib.h>
int a[10][10],d[10][10],n;

int min(int a, int b)
{
   return (a<b) ? a:b;
}

void path()
{
   int i, j, k;
   for(k=0;k<n;k++)
   {
      for(i=0;i<n;i++)
      {
         for(j=0;j<n;j++)
         {
            d[i][j]= min(d[i][j], d[i][k]+d[k][j]);
         }
      }
   }
}
int main()
{
   int i,j;
   printf("Enter the no of Vertices:");
   scanf("%d", &n);
   printf("\nEnter the cost adjacency matrix:\n");
   for(i=0;i<n;i++)
   for(j=0;j<n;j++)
   {
   scanf("%d",&a[i][j]);
   d[i][j]= a[i][j];
   }
   path();
   printf("Final Distance Matrix:\n");
   for(i=0;i<n;i++)
   {
   for(j=0;j<n;j++)
   {
   printf("%5d",d[i][j]);
   }
   printf("\n");
```

```
  }
}
```

b) Warshal's Algorithmn

```
#include<stdio.h>
#include<stdlib.h>
int a[10][10],t[10][10],n;
void path()
{
   int i, j, k;
   for(k=0;k<n;k++)
   {
      for(i=0;i<n;i++)
      {
         for(j=0;j<n;j++)
         {
            if((t[i][j]) || (t[i][k]&& t[k][j]))
               t[i][j] =1;
         }
      }
   }
}
int main()
{
   int i,j;
   printf("Enter the no of Vertices:");
   scanf("%d", &n);
   printf("\nEnter the adjacency matrix:\n");
   for(i=0;i<n;i++)
   for(j=0;j<n;j++)
   {
   scanf("%d",&a[i][j]);
   t[i][j]= a[i][j];
   }
   path();
   printf("Transitive Matrix:\n");
   for(i=0;i<n;i++)
   {
   for(j=0;j<n;j++)
   {
   printf("%5d",t[i][j]);
   }
   printf("\n");
  }
}
```

4. Program in C to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.

```
 #include<stdio.h>
#include<conio.h>
#define infinity 999
void dij(int n,int v,int cost[10][10],int dist[100])
{
```

```
int i,u,count,w,flag[10],min;
for(i=1;i<=n;i++)
flag[i]=0,dist[i]=cost[v][i];
count=2;
while(count<=n)
{
min=99; for(w=1;w<=n;w++)
if(dist[w]<min && !flag[w])
min=dist[w],u=w;
flag[u]=1;
count++;
for(w=1;w<=n;w++)
if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
dist[w]=dist[u]+cost[u][w];
}
}

int main()
{
int n,v,i,j,cost[10][10],dist[10];
printf("\n Enter the number of nodes:");
scanf("%d",&n);
printf("\n Enter the cost matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=infinity;
}
printf("\n Enter the source matrix:");
scanf("%d",&v);
dij(n,v,cost,dist);
printf("\n Shortest path:\n");
for(i=1;i<=n;i++)
if(i!=v)
printf("%d->%d,cost=%d\n",v,i,dist[i]);
}
```

5. Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph.

```
#include<stdio.h>
#include<conio.h>
int a[10][10],n,indegre[10];
void find_indegre()
{ int j,i,sum;
for(j=0;j<n;j++)
{
sum=0;
for(i=0;i<n;i++)
sum+=a[i][j];
```

```
indegre[j]=sum;
}
}
void topology()
{
int i,u,v,t[10],s[10],top=-1,k=0;
find_indegre();  for(i=0;i<n;i++)
{
if(indegre[i]==0)  s[++top]=i;
}
while(top!=-1)
{
u=s[top--];
t[k++]=u;
for(v=0;v<n;v++)
{
if(a[u][v]==1)
{
indegre[v]--;
if(indegre[v]==0) s[++top]=v;
}
}
}
printf("The topological Sequence is:\n");
for(i=0;i<n;i++)
printf("%d ",t[i]);
}

int main()
{
int i,j;
printf("Enter number of jobs:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
}
topology();
}
```

6. C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.

```
#include<stdio.h>
#include<conio.h>
int w[10],p[10],v[10][10],n,i,j,cap,x[10]={0};
int max(int i,int j)
{
return ((i>j)?i:j);
}
int knap(int i,int j)
```

```
{
int value;
if(v[i][j]<0)
{
if(j<w[i])
value=knap(i-1,j);
else
value=max(knap(i-1,j),p[i]+knap(i-1,j-w[i]));
v[i][j]=value;
}
return(v[i][j]);
}

int main()
{
int profit,count=0;
printf("\nEnter the number of elements\n");
scanf("%d",&n);
printf("Enter the profit and weights of the elements\n");
for(i=1;i<=n;i++)
{
printf("For item no %d\n",i);
scanf("%d%d",&p[i],&w[i]);
}
printf("\nEnter the capacity \n");
scanf("%d",&cap);
for(i=0;i<=n;i++)
for(j=0;j<=cap;j++)
if((i==0)||(j==0))
v[i][j]=0;
else
v[i][j]=-1;
profit=knap(n,cap);
i=n;
j=cap;
while(j!=0&&i!=0)
{
if(v[i][j]!=v[i-1][j])
{ x[i]=
1;
j=j-w[i];
i--;
}
else
i--;
}
printf("Items included are\n");
printf("Sl.no\tweight\tprofit\n");
for(i=1;i<=n;i++)
if(x[i])
printf("%d\t%d\t%d\n",++count,w[i],p[i]);
printf("Total profit = %d\n",profit);
}
```

7. C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

```
#include<stdio.h>
#include<stdlib.h>
int main()
 {
    float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
    int n,i,j;
    printf("Enter the number of items :");
    scanf("%d",&n);
   for (i = 0; i < n; i++)
   {
      printf("Enter Weight and Profit for item[%d] :\n",i);
      scanf("%f %f", &weight[i], &profit[i]);
   }
   printf("Enter the capacity of knapsack :\n");
   scanf("%f",&capacity);

    for(i=0;i<n;i++)
       ratio[i]=profit[i]/weight[i];

   for (i = 0; i < n; i++)
     for (j = i + 1; j < n; j++)
        if (ratio[i] < ratio[j])
        {
           temp = ratio[j];
           ratio[j] = ratio[i];
           ratio[i] = temp;

           temp = weight[j];
           weight[j] = weight[i];
           weight[i] = temp;

           temp = profit[j];
           profit[j] = profit[i];
           profit[i] = temp;
        }

    printf("Knapsack problems using Greedy Algorithm:\n");
    for (i = 0; i < n; i++)
    {
     if (weight[i] > capacity)
        break;
      else
     {
        Totalvalue = Totalvalue + profit[i];
        capacity = capacity - weight[i];
      }
    }
      if (i < n)
      Totalvalue = Totalvalue + (ratio[i]*capacity);
```

```
    printf("\nThe maximum value is :%f\n",Totalvalue);
    return 0;
}
```

8. C Program to find a subset of a given set S = {sl , s2,..... ,sn} of n positive integers whose sum is equal to a given positive integer d.

```
#include<stdio.h>
#include<conio.h>
int s[10] , x[10],d ;
void sumofsub ( int , int , int ) ;
int main ()
{
int n , sum = 0 ;
int i ;
printf ( " \n Enter the size of the set : " ) ;
scanf ( "%d" , &n ) ;
printf ( " \n Enter the set in increasing order:\n" ) ;
for ( i = 1 ; i <= n ; i++ )
scanf ("%d", &s[i] ) ;
printf ( " \n Enter the value of d : \n " ) ;
scanf ( "%d" , &d ) ;
for ( i = 1 ; i <= n ; i++ )
sum = sum + s[i] ;
if ( sum < d || s[1] > d )
printf ( " \n No subset possible : " ) ;
else
sumofsub ( 0 , 1 , sum ) ;
}
void sumofsub ( int m , int k , int r )
{
int i=1 ;
x[k] = 1 ;
if ( ( m + s[k] ) == d )
{
printf("Subset:");
for ( i = 1 ; i <= k ; i++ )
if ( x[i] == 1 )
printf ( "\t%d" , s[i] ) ;
printf ( "\n" ) ;
}
else
if ( m + s[k] + s[k+1] <= d )
sumofsub ( m + s[k] , k + 1 , r - s[k] ) ;
if ( ( m + r - s[k] >= d ) && ( m + s[k+1] <=d ) )
{
x[k] = 0;
sumofsub ( m , k + 1 , r - s[k] ) ;
}
}
```

9. Design and implement C Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

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

void selection(int arr[], int n)
{
   int i, j, small;

   for (i = 0; i < n-1; i++)     // One by one move boundary of unsorted subarray
   {
      small = i; //minimum element in unsorted array

      for (j = i+1; j < n; j++)
      if (arr[j] < arr[small])
         small = j;
// Swap the minimum element with the first element
   int temp = arr[small];
   arr[small] = arr[i];
   arr[i] = temp;
   }
}

int main()
{
int n, a[1000],k;
clock_t st,et;
double ts;
printf("\n Enter How many Numbers: ");
scanf("%d", &n);
printf("\nThe Random Numbers are:\n");
for(k=1; k<=n; k++)
 {
a[k]=rand();
printf("%d\t",a[k]);
 }
st=clock();
selection(a,n);
et=clock();
ts=(double)(et-st)/CLOCKS_PER_SEC;
printf("\nSorted Numbers are: \n ");
for(k=1; k<=n; k++)
printf("%d\t", a[k]);
printf("\nThe time taken is %e",ts);
}
```

10. Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

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

```
void Exch(int *p, int *q)
{
int temp = *p;
*p = *q;
*q = temp;
}
void QuickSort(int a[], int low, int high)
{
int i, j, key, k;
if(low>=high)
return;
key=low; i=low+1; j=high;
while(i<=j)
{
while ( a[i] <= a[key] ) i=i+1;
while ( a[j] > a[key] ) j=j-1;
if(i<j) Exch(&a[i], &a[j]);
}
Exch(&a[j],    &a[key]);
QuickSort(a,  low,  j-1);
QuickSort(a, j+1, high);
}
int main()
{
int n, a[1000],k;
clock_t st,et;
double ts;
srand(time(NULL));
printf("\n Enter How many Numbers: ");
scanf("%d", &n);
printf("\nThe Random Numbers are:\n");
for(k=1; k<=n; k++)
 {
a[k]=rand();
printf("%d\t",a[k]);
 }
st=clock();
QuickSort(a, 1, n);
et=clock();
ts=(double)(et-st)/CLOCKS_PER_SEC;
printf("\nSorted Numbers are: \n ");
for(k=1; k<=n; k++)
printf("%d\t", a[k]);
printf("\nThe time taken is %e",ts);
}
```

11. Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

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

```
#include<time.h>

void Merge(int a[], int low, int mid, int high)
{
int i, j, k, b[20];
i=low; j=mid+1; k=low;
while ( i<=mid && j<=high )
{
if( a[i] <= a[j] )
b[k++] = a[i++] ;
else
b[k++] = a[j++] ;
}
while (i<=mid) b[k++] = a[i++] ;
while (j<=high) b[k++] = a[j++] ;
for(k=low; k<=high; k++)
a[k] = b[k];
}

void MergeSort(int a[], int low, int high)
{
int mid;
if(low >= high)
return;
mid = (low+high)/2 ;
MergeSort(a, low, mid);
MergeSort(a, mid+1, high);
Merge(a, low, mid, high);
}

int main()
{
int n, a[2000],k;
clock_t st,et;
double ts;
srand(time(NULL));
printf("\n Enter How many Numbers:");
scanf("%d", &n);
printf("\nThe Random Numbers are:\n");
for(k=1; k<=n; k++)
{
a[k]=rand();
printf("%d\t", a[k]);
}
st=clock();
MergeSort(a, 1, n);
et=clock();
ts=(double)(et-st)/CLOCKS_PER_SEC;
printf("\n Sorted Numbers are : \n ");
for(k=1; k<=n; k++)
printf("%d\t", a[k]);
printf("\nThe time taken is %e",ts);
}
```

12. Design and implement C/C++ Program for N Queen's problem using Backtracking.

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

int a[30],count=0;
int place(int pos)
{
int i;
for(i=1;i<pos;i++)
{
if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
return 0;
}
return 1;
}

void print_sol(int n)
{
int i,j;
count++;
printf("\n\nSolution #%d:\n",count);
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
if(a[i]==j)
printf("Q\t");
else
printf("*\t");
}
printf("\n");
}
}

void queen(int n)
{
int k=1;
a[k]=0;
while(k!=0)
{
a[k]=a[k]+1;
while((a[k]<=n)&&!place(k))
a[k]++;
if(a[k]<=n)
{
if(k==n)
print_sol(n);
else
{k+
+;
a[k]=0;
```

```
}
}
else
k--;
}
}

int main()
{
int i,n;
printf("Enter the number of Queens\n");
scanf("%d",&n);
queen(n);
printf("\nTotal solutions=%d",count);
}
```