DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY

(Common to CSE & ISE)

Subject Code: 10CSL47	**I.A. Marks : 25**
Hours/Week : 03	**Exam Hours: 03**
Total Hours : 42	**Exam Marks: 50**

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

1. Sort a given set of elements using the Quicksort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.
   The elements can be read from a file or can be generated using the random number generator.

2. Using OpenMP, implement a parallelized Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

3. a. Obtain the Topological ordering of vertices in a given digraph.
   b. Compute the transitive closure of a given directed graph using Warshall's algorithm.

4. Implement 0/1 Knapsack problem using Dynamic Programming.

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

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

7. 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.

8. 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. 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.

9. Implement any scheme to find the optimal solution for the Traveling Salesperson problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.

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

11. Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. Parallelize this algorithm, implement it using OpenMP and determine the speed-up achieved.

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

Note: In the examination *each* student picks one question from the lot of *all* 12 questions.

1. Sort a given set of elements using the Quicksort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
int partition(int a[],int low,int high)
{
        int i,j,key,temp;
        key=a[low];
        i=low+1;
        j=high;
        while(1)
        {
                while(i<high && key>=a[i ])
                        i++;
                while(key<a[j])
                        j--;
                if(i<j)
                {
                        temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
                }
                else
                {
                        temp=a[low];
                        a[low]=a[j];
                        a[j]=temp;
                        return j;
                }
        }
}

void quicksort(int a[],int low,int high)
{
        int j;
        if(low<high)
        {
                j=partition(a,low,high);
                quicksort(a,low,j-1);
                quicksort(a,j+1,high);
        }
}

void main()
{
        int i,n,a[20];
        clock_t st,end;
        clrscr();
        printf("\n enter the valu of n");
        scanf("%d",&n);
        printf("\n enter no to be sorted");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);

        st=clock();
        quicksort(a,0,n-1);
        end=clock();

        printf("\n the sorted array is\n");
        for(i=0;i<n;i++)
                printf("%d\n",a[i]);

        printf("\n time taken=%d",end-st/CLK_TCK);
        getch();
}
```

2. Using OpenMP, implement a parallelized Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
#include<omp.h>
#include<stdio.h>
#include<time.h>
void merge(int a[],int low,int mid,int high)
{
        int c[10000],i,j,k;
        i=low;
        j=mid+1;
        k=low;]
        {
                while((i<=mid)&&(j<=high))
                {
                        if(a[i]<=a[j])
                                c[k++]=a[i++];
                        else
                                c[k++]=a[j++];
                }
                while(i<=mid)
                        c[k++]=a[i++];
                while(j<=high)
                        c[k++]=a[j++];
                for(i=low;i<=high;i++)
                        a[i]=c[i];
        }
}

void mergesort(int a[],int low,int high)
{
        int mid;
        if(low<high)
        {
                #pragma omp parallel sections num_threads(5)
                {
                        mid=(low+high)/2;
                        #pragma omp section
                        mergesort(a,low,mid);
                        #pragma omp section
                        mergesort(a,mid+1,high);
                        #pragma omp section
                        merge(a,low,mid,high);
                }
        }
}

int main()
{
        int a[10000],i,n;
        clock_t st,end;
        printf("enter number of elements to sort\n");
        scanf("%d",&n);
        for(i=0;i<n;i++)
                a[i]=rand()%100;
        printf("the elements are\n",n);
        for(i=0;i<n;i++)
                printf("%d\n",a[i]);
        st=clock();
        mergesort(a,0,n-1);
        end=clock();
        printf("\n sorted elements are\n");
        for(i=0;i<n;i++)
                printf("%d\n",a[i]);
        printf("\n time taken=%f    \n",(end-st)/(double)CLOCKS_PER_SEC);

}
```

3. a. Obtain the Topological ordering of vertices in a given digraph.

```
#include<stdio.h>
#include<conio.h>
void read_adjacency_matrix(int n, int a[10][10])
{
        int i,j;
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
}

void find_ind(int n,int a[10][10],int ind [])
{
        int i,j,sum;
        for(j=0;i<n;i++)
        {
                sum=0;
                for(i=0;i<n;i++)
                        sum+=a[i][j];
                ind[j]=sum;
        }
}

void topological_sort(int n, int a[10][10])
{
        int i,v,k,u,top,t[10],ind[10],s[10];
        find_ind(n,a,ind);
        top=-1;
        k=0;
        for(i=0;i<n;i++)
        {
                if(ind[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)
                        {
                                ind[v]--;
                                if(ind[v]==0)
                                {
                                        s[++top]=v;
                                }
                        }
                }
        }
        printf("the topological sequence\n");
        for(i=0;i<n;i++)
                printf("%d\n",t[i]);
}

void main()
{
        int n,a[10][10];
        clrscr();
        printf("enter the no of nodes\n");
        scanf("%d",&n);
        printf("enter the adjacency matrix\n");
        read_adjacency_matrix(n,a);
        topological_sort(n,a);
}
```

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

```
#include<stdio.h>
#include<conio.h>
void warshall(int n,int a[10][10],int p[10][10])
{
        int i,j,k;
        for(i=0;i<n;i++)                              //make a copy of adjacency matrix
                for(j=0;j<n;j++)
                        p[i][j]=a[i][j];

        for(k=0;k<n;k++)                              //find the transitive closure [path matrix]
                for(i=0;i<n;i++)
                        for(j=0;j<n;j++)
                                if(p[i][j]==0 && (p[i][k]==1 && p[k][j]==1))
                                        p[i][j]=1;

}

void read_matrix(int n,int a[10][10])
{
        int i,j;
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&a[i][j]);
}

void write_matrix(int n,int a[10][10])
{
        int i,j;
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                printf("%d\t",a[i][j]);
                printf("\n");
        }
}


void main()
{
        int n,a[10][10],p[10][10];
        clrscr();
        printf("enter the no of nodes\n");
        scanf("%d",&n);
        printf("enter the cost adjecency matrix\n");
        read_matrix(n,a);

        warshall(n,a,p);

        printf("transitive closure is as shown below\n");
        write_matrix(n,p);
        getch();
}
```

4. Implement 0/1 Knapsack problem using Dynamic Programming.

```
#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]);
}

void main()
{
        int profit,count=0;
        clrscr();
        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);
        getch();
}
```

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

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

void dij(int n,int v,int cost[10][10],int dist[])
{
        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];
        }
}

void main()
{
        int n,v,i,j,cost[10][10],dist[10];
        clrscr();
        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]=999;
                }
        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]);
        getch();
}
```

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

```
#include<stdio.h>
#include<conio.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 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;
}

void main()
{
        clrscr();
        printf("\nEnter the no. of vertices\n");
        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("\nThe edges of Minimum Cost Spanning Tree are\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;
                                }
                        }
                }
                u=find(u);
                v=find(v);
                if(uni(u,v))
                {
                        printf("\n%d edge (%d,%d) =%d\n",ne++,a,b,min);
                        mincost=mincost+min;
                }
                cost[a][b]=cost[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);
        getch();
}
```

7. a. Print all the nodes reachable from a given starting node in a digraph using BFS method.

```
#include<stdio.h>
#include<process.h>
void bfs(int n,int a[10][10],int source,int s[])
{

        int f,r,q[20];
        int u,v;
        int i;
        for(i=0;i<n;i++)
                s[i]=0;                         //initially, no node is visited
        f=r=0;
        q[r]=source;                            //insert 'source' into Q
        s[source]=1;                            //add source to 's'[mark source as visited]

        while(f<=r)                         //as long as queue is not empty
        {
                u=q[f++];                           //delete the next vertex to be explored
                for(v=0;v<n;v++)            //find the nodes v which are adjacent to u
                {
                        if(a[u][v]==1 && s[v]==0)
                        {
                                s[v]=1;         //add v to 's' indicates that v is visited now
                                q[++r]=v;       //insert new vertex into Q for exploration
                        }
                }
        }
}

void main()
{
        int n,i,j,source,a[10][10],s[10],flag;
        clrscr();
        printf("enter number of vertice\n");
        scanf("%d",&n);
        printf("enter adjacency matrix\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&a[i][j]);

        printf("enter source\n ");
        scanf("%d",&source);

        bfs(n,a,source,s);

        for(i=0;i<n;i++)
        {
                if(s[i]==0)
                        printf("%d is not reachable\n",i);
                else
                        printf("%d is reachable\n",i);
        }
        getch();
}
```

7.b. Check whether a given graph is connected or not using DFS method.

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

int n.s[10],a[10][10],flag;

void dfs(int u,int s[])
{
        int v;
        s[u]=1;
        for(v=0;v<n;v++)
                if(a[u][v]==1 && s[v]==0)
                        dfs(v,s);
}

int connectivity(int n)
{
        int i,j;
        for(j=0;j<n;j++)
        {
                for(i=0;i<n;i++)
                        s[i]=0;
                dfs(j,s);
                flag=0;
                for(i=0;i<n;i++)
                        if(s[i]==0)
                                flag=1;
                if(flag==0)
                        return 1;
        }
        return 0;
}

void main()
{

        int i,j,n;
        clrscr();
        printf("enter n value");
        scanf("%d",&n);
        printf("enter adjajency matrix");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&a[i][j]);
        flag=connectivity();
        if(flag==1)
                printf("connected");
        else
                printf("not connected");
        getch();

 }
```

8. 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.

```
#include<stdio.h>
#include<conio.h>
#define max 10
int s[max],x[max];
int d;
void sumofsub(int,int,int);
void main()
{
        int n,sum=0;
        int i;
        clrscr();
        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
        {
                printf("possible subsets are as follows\n");
                sumofsub(0,1,sum);
        }

        getch();
}

void sumofsub(int m,int k,int r)
{
        int i;
        x[k]=1;
        if((m+s[k])==d)
        {
                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. Implement any scheme to find the optimal solution for the Traveling Salesperson problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int a[10][10],visited[10],n,cost=0;
void get()
{
        int i,j;
        printf("\n\nEnter Number of Cities: ");
        scanf("%d",&n);
        printf("\nEnter Cost Matrix: \n");
        for( i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                        scanf("%d",&a[i][j]);
                visited[i]=0;
        }
}

void mincost(int city)
{
        int i,ncity;
        visited[city]=1;
        printf("%d ===> ",city);
        ncity=least(city);
        if(ncity==999)
        {
                ncity=1;
                printf("%d",ncity+1);
                cost=cost+a[city][ncity];
                return;
        }
        mincost(ncity);
}

int least(int c)
{
        int i,nc=999;
        int min=999,kmin;
        for(i=1;i<=n;i++)
        {
                if((a[c][i]!=0)&&(visited[i]==0))
                        if(a[c][i]<min)
                        {
                                min=a[i][0]+a[c][i];
                                kmin=a[c][i];
                                nc=i;
                        }
        }
        if(min!=999)
                cost=cost+kmin;
        return nc;
}


void put()
{
        printf("\n\nMinimum cost:");
        printf("%d",cost);
}

void main()
{
        get();
        printf("\n\nThe Path is:\n\n");
        mincost(1);
        put();
}
```

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

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
        clrscr();
        printf("\n Enter the number of nodes:");
        scanf("%d",&n);
        printf("\n Enter the 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;
                }
        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];
                                                a=u=i;
                                                b=v=j;
                                        }
                if(visited[u]==0 || visited[v]==0)
                {
                        printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                        mincost+=min;
                        visited[b]=1;
                }
                cost[a][b]=cost[b][a]=999;
        }
        printf("\n Minimun cost=%d",mincost);
        getch();
}
```

11. Implement All-Pairs Shortest Paths Problem using Floyd s algorithm. Parallelize this algorithm, implement it using OpenMP and determine the speed-up achieved.

```
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>
#define INF 999
int cost[10][10], a[10][10];

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

void allpaths(int cost[10][10], int a[10][10], int n)
{
        int i, j, k;

        #pragma omp parallel for num_threads(2)
        for(i = 1; i <= n; i++)
        {
                for(j = 1; j <= n; j++)
                        a[i][j] = cost[i][j];
         }
        #pragma omp parallel for num_threads(3)
        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]);
                         }
                 }
         }
}

int main()
{
        int i, j, n;

        printf("\n\nEnter the no. of the 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]);
        allpaths(cost, a, n);
        printf("The shortest poath obtained is as follows: \n");
        for(i = 1; i <= n; i++)
        {
                for(j = 1; j <= n; j++)
                        printf("%d\t", a[i][j]);
                printf("\n");
        }
        return 0;
}
```

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

```
#include<stdio.h>
int place(int x[],int k)
{
        int i;
        for(i=1;i<k;i++)            //check whether 2 queens attack vertically or horizontall
                if(x[i]==x[k] || (abs(x[i]-x[k])==abs(i-k)))
                        return 0;    //queen cannot be placed in the kth column

        return 1;                            //kth queen can be successfully placed
}


void display(int x[],int n)
{
        char board[20][20];
        int i,j;
        for(i=1;i<=n;i++)                    // no queen has been placed initailly
                for(j=1;j<=n;j++)
                        board[i][j]='x';

        for(i=1;i<=n;i++)                    // place the queens on the chess board
                board[i][x[i]]='Q';

        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                        printf("%c  ",board[i][j]);
                printf("\n");
        }
        printf("\n");
 }



void nqueens(int n)
{
        int x[30];
        int k=1;
        x[k]=0;      //initialize row


        while(k!=0)
        {
                x[k]=x[k]+1;                        //go to next column
                while((x[k]<=n) && !place(x,k))
                        x[k]=x[k]+1;
                if(x[k]<=n)                //place the queen
                        if(k==n)
                                display(x,n);
                        else
                        {
                                k++;                //try for next queen
                                x[k]=0;
                        }
                else
                        k--;                        // backtrack, queen can't be placed
        }
}

void main()
{
        int n;
        clrscr();
        printf("enter number of queens n>3");
        scanf("%d", &n);
        printf("the solution to queen problem is:\n");
        nqueens(n);
}
```